# Comparative Study of Computational Methods for the Solution of the Two-Dimensional Reservoir Flow Equations

by

Hassan Saeed Al-Towailib

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

**MASTER OF SCIENCE**

In

**PETROLEUM ENGINEERING**

February, 1993

Order Number 1355321

# Comparative study of computational methods for the solution of the two-dimensional reservoir flow equations

Al-Towailib, Hassan Saeed, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

# COMPARATIVE STUDY OF COMPUTATIONAL METHODS FOR THE SOLUTION OF THE TWO-DIMENSIONAL RESERVOIR FLOW EQUATIONS

BY

HASSAN SAEED AL-TOWAILIB

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

**MASTER OF SCIENCE**

In

PETROLEUM ENGINEERING

FEBRUARY 1993

**KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS**

**DHAHRAN, SAUDI ARABIA**

**COLLEGE OF GRADUATE STUDIES**

This thesis, written by Hassan Saeed Al-Towailib under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE in petroleum engineering.

Thesis Committee

M. A. Marhoun

Dr. Muhammad Al-Marhoun
Thesis Advisor

Dr. Ala H. Al-Rabeh
Member

H. Y. Al-Yousef

Dr. Hassan Y. Al-Yousef
Member

Dr. Khalid Al-Fossail
Department Chairman

Dr. Ala H. Al-Rabeh
Dean, College of Graduate Studies

10th of May 1993

Date

# DEDICATION

**To My Loving Parents and Family**

# ACKNOWLEDGEMENTS

Acknowledgement is due to King Fahd University of Petroleum and Minerals for support of this research.

I wish to express my appreciation to Professor M. A. AL-Marhoun who served as my major advisor. Thanks are due to the members of my Thesis Committee Dr. A. A. Al-Rabeh and Dr. H. Y. Al-Yousef for their keen interest and valuable suggestions. I also wish to thank Dr. A. Haroun̲aka for his review of the final draft of the thesis.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# خلاصة الرسالة

اسم الطالـــب : حسن سعيد عباس الطويلب
عنوان الرسالة : دراسة مقارنة للطرق الحسابية لحل معادلات التدفق في المكامن الثنائية الابعاد
التـخصـــص : هندسة البترول
تاريخ الشهادة : فبراير ١٩٩٣ م

ظهرت عدة طرق مـختلفة لحل منظومـات المعادلات الجبرية الناتجة عن الحل التقريبي لمعادلات التدفق في المكامن الثنائية الابعاد. تلك الطرق تحل منظومات المعـادلات الجـبـرية إمـا مـبـاشـرة او بطريقـة المحـاولات التـصـوييـبـيـة . الرسالة تشـمل دراسـة مـقـارنة جديدة لأكثـر تلك الطرق اسـتـخداماً . هذه الدراسـة تتـضـمـن تحليـلاً وافياً لكل طريقة بالاضافة الى مـقارنة الفعاليات الحسابية عبر الوقت الحسابي وحيز التخزين المطلوبين . وتحتوي ايضاً على شـرح لبعض الاساليب المستـخدمـة في ترتيب منظومـات المعـادلات الجبرية . استخدم في المقارنة مكمن مغمور بالماء متجانس ذو طورين (زيت – ماء) وقد اسـتـعـمل مـحـاكي مكامن رياضـي بسـيط ذو طورين وثلاثي الابعـاد. وقد اظهرت النتائج أن طريقة د٤- في ترتيب المعادلات هي الاكثر فعالية بين الطرق المباشرة بينما طريقة التزايد التعاقبي بالنقاط كانت الاكثر فعالية من بين طرق المحاولات التصويبية وكانت طريقة التحليل الى عوامل متداخلة هي الاسرع للوصول للحل من طرق المحاولات التصويبية.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران، المملكة العربية السعودية
فبراير ١٩٩٣ م

# THESIS ABSTRACT


**STUDENT NAME** : Hassan S. Al-Towailib


**TITLE OF STUDY** : **Comparative Study of Computational Methods for the Solution of the Two-Dimensional Reservoir Flow Equations**

**MAJOR FIELD** : Petroleum Engineering

**DATE OF DEGREE** : February 1993


Several methods have been developed for solving the system of linear equations that arise in approximating the solution of two-dimensional reservoir flow equations. Those methods solve the system of linear equations either directly or iteratively.

In this thesis a comparative study of the most commonly used methods is made. The study includes a full analysis of each method plus a comparison of the computing efficiencies in terms of both the computation times and the memory space requirements. It also includes a discussion of grid ordering techniques that have appeared in the literature.

The test problem used in the comparison is a homogeneous, two-phase (oil-water), two-dimensional five-spot water flooding. A simple two-phase three dimensional reservoir simulator was used. The results showed **D4-Gaussian Elimination** method to be the most efficient of the direct methods while the **Point Successive Over-relaxation** method was the most efficient among the iterative techniques. **Nested Factorization** method had the fastest convergence of all the iterative techniques.



**MASTER OF SCIENCE DEGREE**
**KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS**
**Dhahran, Saudi Arabia**
**February 1993**

# CHAPTER 1

# INTRODUCTION

The efficiency of a reservoir simulator as a working and economical tool hinges upon the ability of its algorithms to solve the fluid flow equations accurately and efficiently. The computational time can be expensive; therefore, a comparative study of the available methods that aims at identifying the most efficient and accurate method for a given reservoir problem can be beneficial.

A variety of methods have been developed for solving the large sets of algebraic equations that arise in approximating the solution of multi-dimensional partial differential equations by either explicit or implicit techniques. A difference equation is written for each grid point in the region of interest, and the resulting set of simultaneous algebraic equations is solved for each time step. A comparative study of the available methods provides some insight into the relation between how well the system of flow equations is solved and the resulting "quality" of the solutions.

The solution of the system of algebraic equations can be accomplished by direct or iterative methods. The most commonly used direct methods to solve those sets of equations employ the process of Gaussian elimination. This approach is the most efficient method available for small sets of equations but not for large

sets since their required storage and computer time have been found to increase drastically with slight increases in the size of the matrices. Direct solutions have been found to be more reliable, but they normally require a larger storage space and more computer time than the iterative methods. The choice of the solution technique will generally depend on the nature and the size of the reservoir being modeled.

The ordering of the grid system has been found to play a big role in the effectiveness of the direct methods. Different orderings of the grid system cause the coefficient matrix of the linear system of equations to take different forms. Various ordering techniques have appeared in the literature. The standard ordering, in which the points of a three-dimensional grid is first numbered along the shortest direction- i.e., the dimension with the fewest number of grid points- then in the next shortest direction, and finally in the longest direction, was found to require large computer storage and overhead work to store and generate the solution code compared to other ordering schemes such as those that were developed by Price and Coats [9]. Al-Marhoun [10] proposed another ordering technique. In this technique each element along different secondary diagonals in the coefficient matrix was re-arranged. It was called restricted alternating diagonal ordering ( **RAD** ).

It is often more economical in terms of computing time and storage to use an iterative technique rather than a direct technique. Iterative methods usually require a set of parameters to accelerate their convergence to an acceptable level. Some examples of the available iterative methods which are in common use in reservoir simulation that are of this category are : the Successive Over-Relaxation methods, Point **SOR** and Line **SOR**, the Alternating Direction Implicit Procedure, **ADIP**, and the Strongly Implicit Procedure, **SIP.** Some of the latest

iterative methods do not require any iteration parameter. Two examples of these are the Conjugate Gradient-Truncated Direct method, **CGTD,** and Nested Factorization, **NF**.

The objective of this thesis is to compare the most commonly used computational solution techniques in reservoir simulation. This study will be limited to solving the equations governing a two-dimensional reservoir fluid flow. In this study a full analysis and comparison of some commonly used computational techniques and their computing efficiencies in terms of both the computation times and the memory space requirements will be made.

# CHAPTER 2

# LITERATURE REVIEW

Comparative studies of computational techniques in reservoir simulation that have appeared in the literature are presented in the following sections.

In 1966, Coats et al. [1] compared the alternating direction explicit procedure, **ADEP**, versus the alternating direction implicit procedure, **ADIP**, in numerical solution of some examples of applications of reservoir fluid flow. They found that **ADEP** is nonconservative in that it fails to preserve no-flow conditions at exterior boundaries, which may cause extremely severe errors in the potential and the material balance, Figures 1 and 2. They concluded that the implicit procedure, **ADIP**, is considerably superior to the explicit procedure, **ADEP**, in terms of accuracy, but it requires about 60 percent more computing time.

In 1967, Stone [2] developed a new computational method, the strongly implicit procedure, **SIP**. The new method was compared against three other iterative methods: the Point-Jacobi, the successive over-relaxation methods ( **SOR** ), and the alternating direction implicit procedure ( **ADIP** ). A series of problems that involved heat conduction in a two-dimensional region were solved using the different methods. The results were presented in terms of the computational work and residual. For a homogenous ideal problem the results

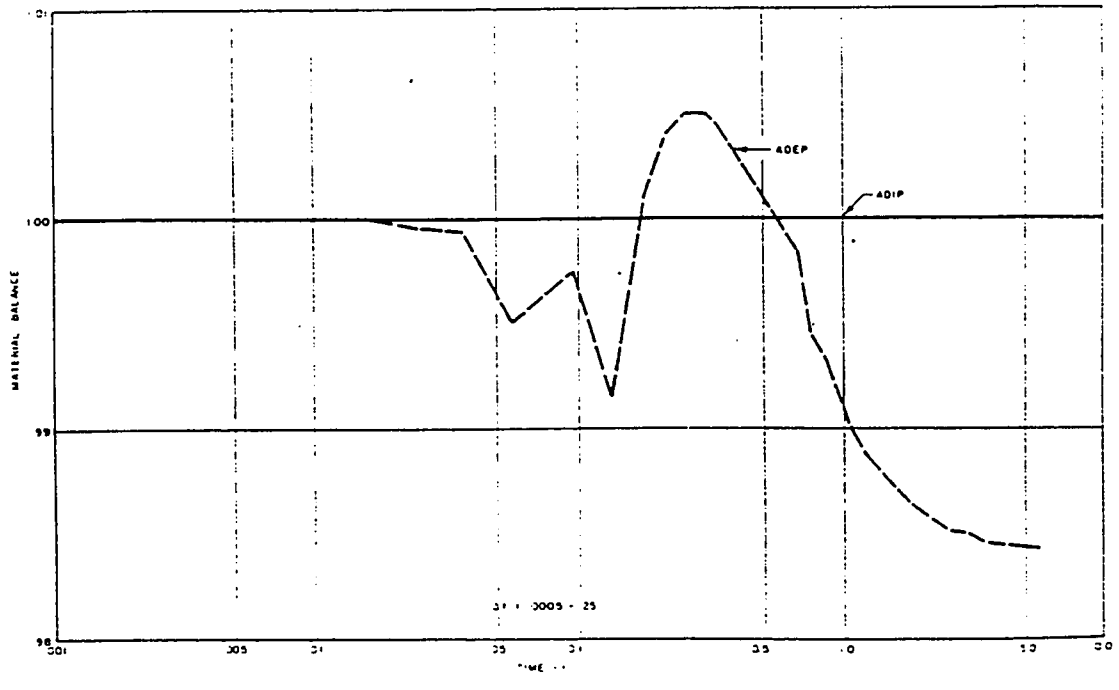



Figure 1 - Comparison of **ADIP** and **ADEP** Material Balances (Well Problem), (Coats et al., 1966)

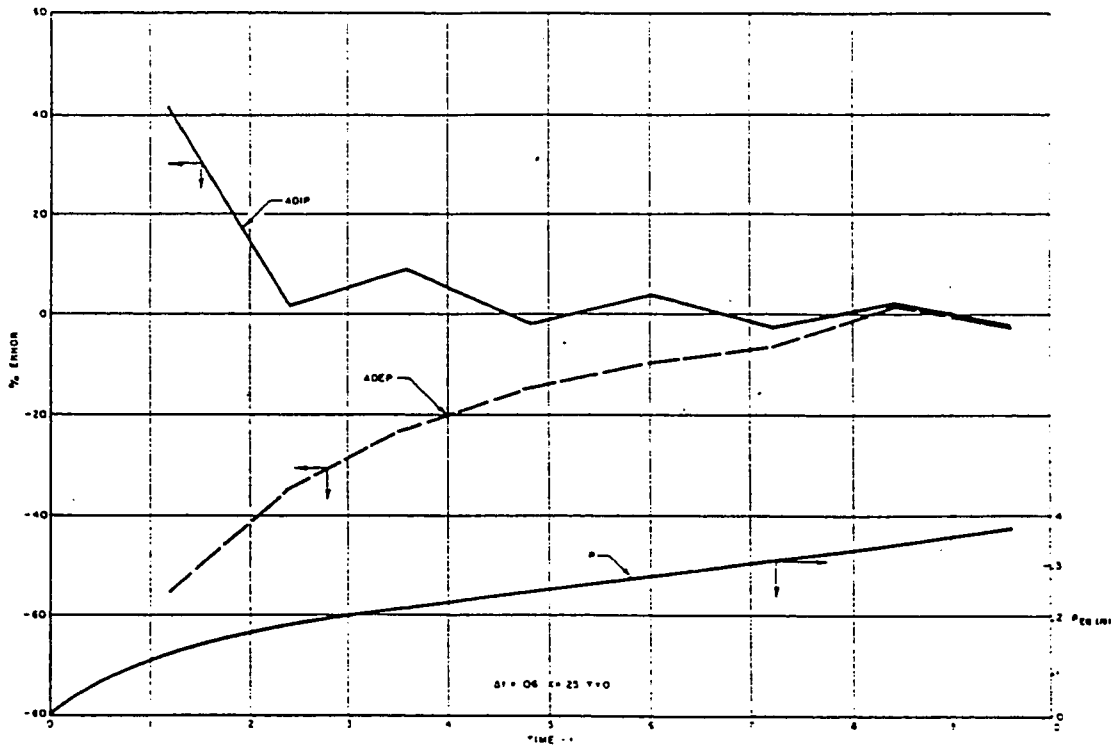



Figure 2 - Comparison of **ADIP** and **ADEP** Errors
(Well Problem ), (Coats et al., 1966)

showed that both **ADIP** and **SIP** converged much faster than the other two methods, Figure 3. **ADIP** was slightly faster than **SIP**, and **SOR** is significantly faster than the Point-Jacobi method. For the heterogeneous model with random permeability variations in subregions, **SIP** maintained faster convergence rate than all of the other methods, Figure 4. Of those three methods, **ADIP** was the most effective. It was also shown that the effectiveness of **SIP** is much less sensitive than **ADIP** to the nature of the problem being solved, see Table 1.

From this study, Stone concluded that **SIP** is moderately sensitive to the nature of the boundary conditions to be applied. Unlike previous methods, this method work requirement is not sensitive to the number of equations to be solved. Moreover, iteration parameters suitable for application may be reliably estimated. It was also concluded that this method requires less computation. For a typical problem involving 961 equations, **SIP** required approximately about one fourth the computation required by the best competitive method.

In 1968, Breitenbach et al. [3] compared the following methods : Gaussian Elimination, **SOR**, and **ADIP** methods. They based their choice for those methods on both the economic adequacy in reservoir simulation, and the usefulness for particular problems. They presented a full analysis in addition to the computational techniques which are vital to actual use of each of those methods. Also, some comparison of typical solution times per time step for various grid sizes was provided. They recommended use of the iterative **ADIP** or **SOR** rather than the Gaussian Elimination for grids over 400 cells. They showed that point successive over-relaxation method, **PSOR**, and the iterative **ADI** method were more efficient than Gaussian Elimination for more than 9x26 cell system, Table 2.

In 1968, Briggs et al. [4] compared the computing times and accuracies in

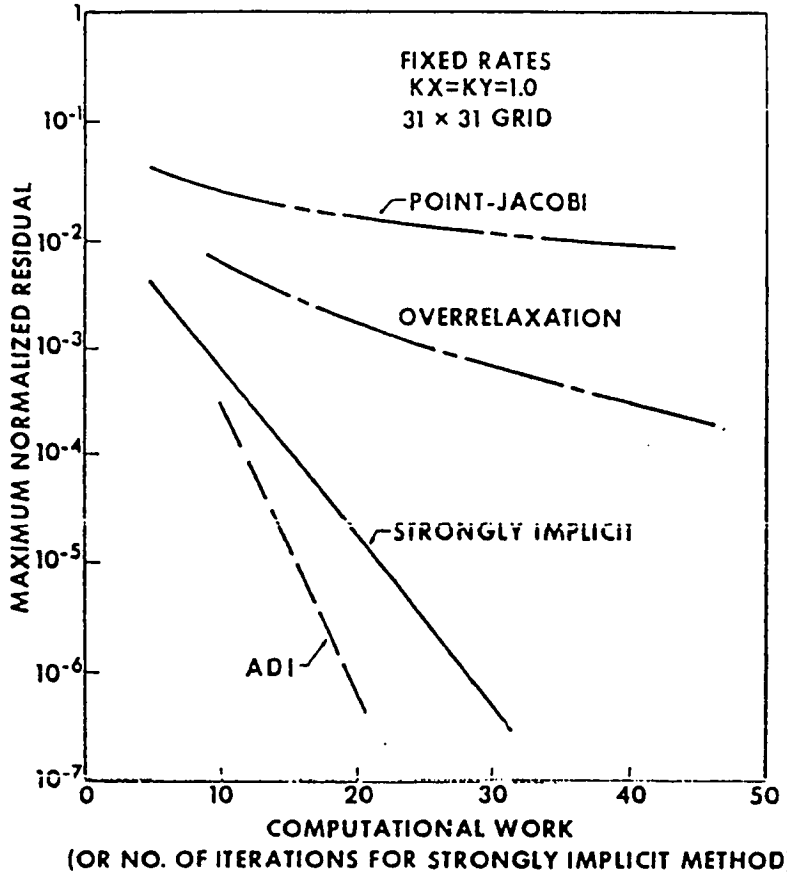



Figure 3 - Comparison of Computational Work Required for Different Iteration Methods - ( Homogenous Model Problem), ( Stone, 1967 )

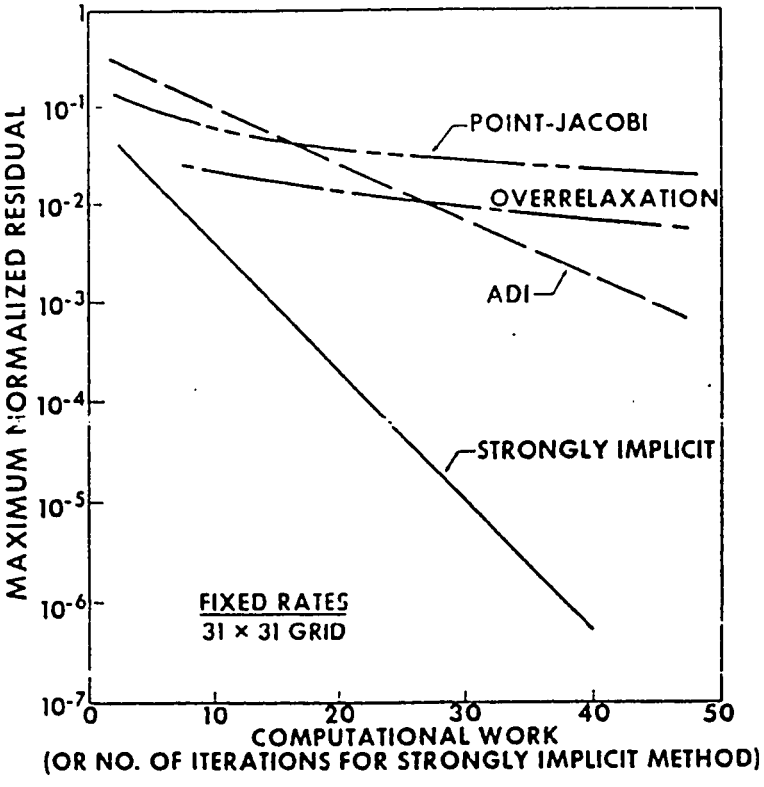



Figure 4 - Comparison of Computational Work Required for Different Iteration Methods - ( Heterogeneous Model Problem), ( Stone, 1967 )

TABLE 1 - Work Requirements of Iterative Methods, (Stone, 1967 )

| Method | Homogeneous Model $k_x = k_y$ | Generalized Model $k_x = 100\ k_y$ | Heterogeneous | |
|---|---|---|---|---|
| | | | Subregions Homogeneous | Random Number Subregions |
| **SIP** | 22 | 16 | 30 | 34 |
| **ADIP** | 16 | 50 | 80 | 127 |
| RATIO : **ADIP/SIP** | 0.73 | ** | 2.66 | 3.74 |

** Omitted because best parameters not used for **ADIP**.

TABLE 2 - Typical Solution Times per Time Step for Various Grid Sizes and for Three Numerical Methods, ( Breitenbach et al., 1968 )

| | Method | | |
|---|---|---|---|
| Grid Size | Gaussian Elimination | SOR | ADIP |
| 3 x 5 | 2 | 5 | 5 |
| 3 x 5 x 2 | 5 | 12 ** | - |
| 9 x 10 | 8 | 11 | - |
| 13 x 11 | 18 | 25 | 19 |
| 9 x 26 * | 21 | 40 ** | - |
| 9 x 26 | 21 | 21 | 22 |
| 26 x 23 | 94 | 59 | 62 |
| 34 x 36 | 302 | 122 | 111 |
| 20 x 24 x 2 | 564 | 110 | - |
| 28 x 51 * | | | |

* Vertical Cross Section
** Actually **LSOR** Method.

solving a model reservoir problem for various grid and time step sizes for **SOR**, Two-line Cyclic Chebyshev Semi-iterative **SOR(2LCC)**, iterative **ADI**, and the non-iterative Peaceman-Rachford (**ADI**) methods. They narrowed their choice of the iterative techniques to **SOR** variants and **ADI** method, since the **SOR** is applicable to a wide variety of problems, while **ADI** is more restricted. However, in many situations where either **SOR** or **ADI** could be applied, **ADI** has a marked advantage in convergence rate. The Two-Line Cyclic Chebyshev (**2LCC**) procedure is a variant of block **SOR**.

They compared the computing times and accuracies in solving a model reservoir problem for various grid and time step sizes for those methods. The model reservoir was rectangular with constant pressure boundaries, and included an injection well. Solutions to the problem were calculated with the above methods with a variety of time-step and grid-spacing sizes. The non-iterative **ADI** produced oscillatory results and was restricted by the time step size ( Figure 5 ). Some estimates of time step sizes sufficient to avoid oscillatory behavior produced by this method were given.

Their results are listed in Tables 3, 4 and 5, and shown in Figures 6, 7 and 8. They concluded that the iterative **ADI** and **2LCC** methods are often superior to the non-iterative **ADI** for reservoir simulation. The **2LCC** is superior to the iterative **ADI** for grids with a small number of computing points, while the iterative **ADI** is better for larger number of points. Also the maximum usable time step size for the non-iterative **ADI** can be predicted accurately.

In 1969, Bjordammen et al. [5] compared the capability and computing efficiency of **SOR** and **ADI** techniques in simulating pressure maintenance by water and gas injection. Several variations of the **SOR** method were used : **PSOR**, point symmetric **SOR**, **LSOR** and line symmetric **SOR**. A total of four reservoir

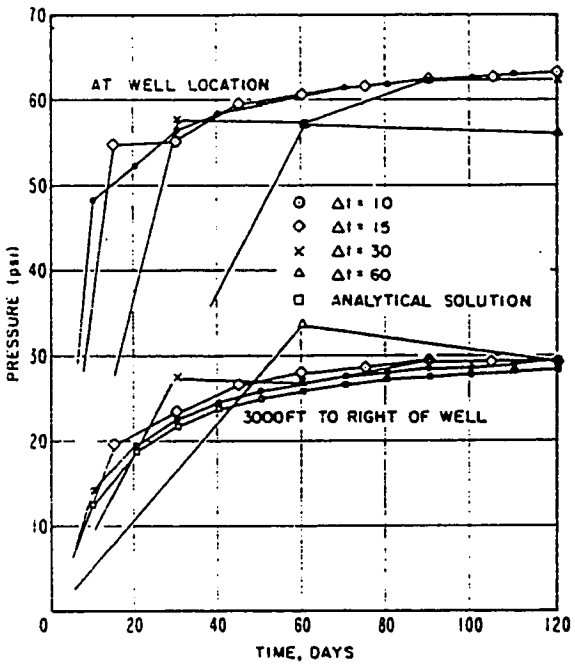



Figure 5 - 231 Point Test Model Results, Non-Iterative ADI, ( Briggs et al., 1968 )

TABLE 3 - Results for 231 Points, ( Briggs et al., 1968 )

| Method | Computing Time (min) | Total Normalized Iterations to Reach 120 Days | Δ t ( Days) |
|---|---|---|---|
| **ADI** | .18 | - | 10 |
| **ADI** | .10 | - | 15 |
| **ADI** | .05 | - | 30 |
| **ADI** | .03 | - | 60 |
| **SOR** | .39 | 187 | 10 |
| **SOR** | .32 | 140 | 15 |
| **SOR** | .20 | 102 | 30 |
| **SOR** | .10 | 59 | 60 |
| **IADI** | .33 | 120 | 10 |
| **IADI** | .27 | 88 | 15 |
| **IADI** | .15 | 48 | 30 |
| **IADI** | .09 | 29 | 60 |
| **2LCC** | .27 | 87 | 10 |
| **2LCC** | .15 | 63 | 15 |
| **2LCC** | .085 | 37 | 30 |
| **2LCC** | .07 | 23 | 60 |

TABLE 4 - Results for 924 Points, $\Delta t$ = 30 Days, ( Briggs et al., 1968 )

| Method | Computing Time (min) | Normalized Iterations to Reach 120 Days |
|---|---|---|
| **ADI** | 0.19 | |
| **SOR** | 0.67 | 143 |
| **IADI** | 0.55 | 51 |
| **2LCC** | 0.51 | 62 |

TABLE 5 - Results for 1891 Points, $\Delta t$ = 30 Days, ( Briggs et al., 1968)

| Method | Computing Time ( min ) | Normalized Iterations to Reach 120 Days |
|---|---|---|
| **ADI** | 0.34 | |
| **SOR** | 2.5 | 237 |
| **IADI** | 1.06 | 56 |
| **2LCC** | 1.5 | 90 |

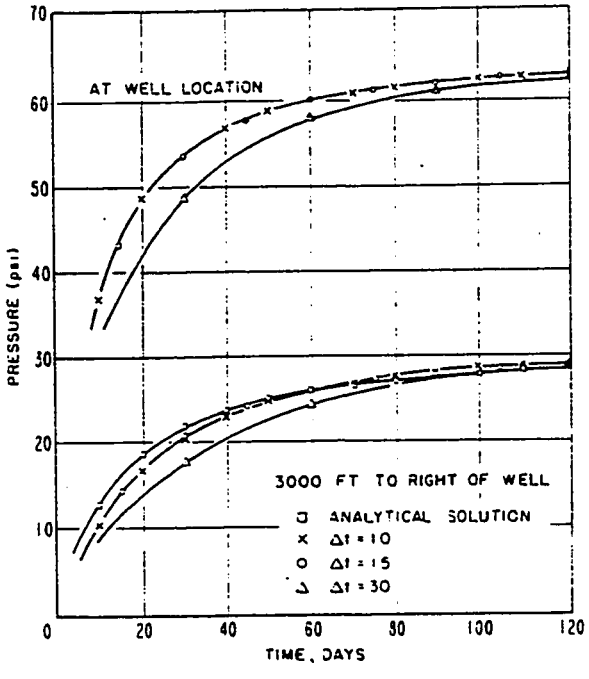



Figure 6 - 231 Point Test Model Results, Iterative Methods, ( Briggs et al., 1968)

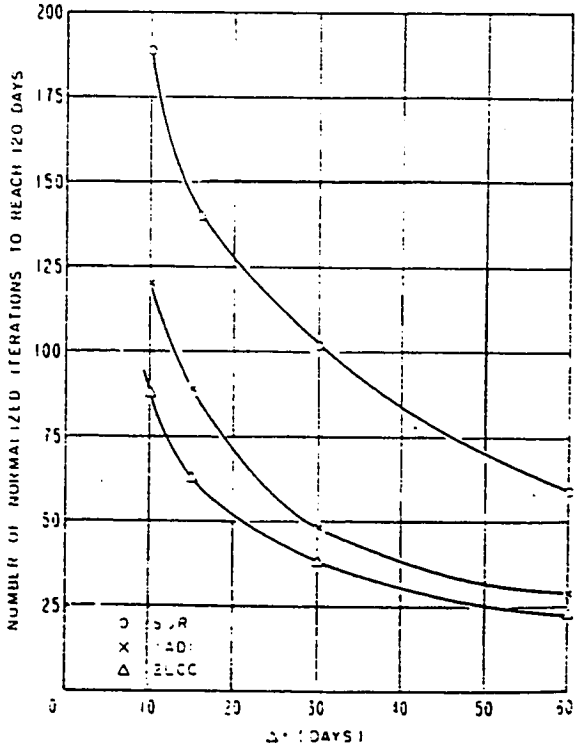



Figure 7 - Number of Normalized Iterations vs Time Step Size for 231 Point Model, (Briggs et al., 1968)

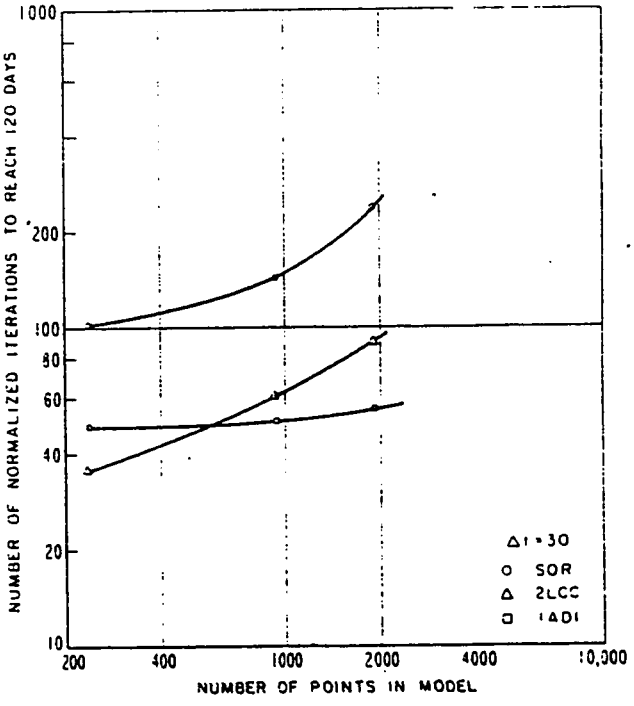



Figure 8 - Number of Normalized Iterations vs Number of Points in Model for Test Model, (Briggs et al., 1968)

problems were simulated, three of which are of the oil-water, pressure maintainance type. Each of those three problems represented the simulation of one-quarter of a 20-acre five-spot waterflood. Two of them are two dimensional and only differed in the grid sizes being employed. The third problem represented a three-layered homogeneous reservoir with injection into three layers and production from the top two layers. The fourth problem was a simulation of gas injection in a cross-section of a slightly tilting homogeneous reservoir.

Their results showed higher convergence rates were obtained with nonsymmetric **SOR** methods as opposed to symmetric ones. Also, higher convergence rates were obtained for **LSOR** as opposed to **PSOR**. The **ADI** method was superior to all the **SOR** techniques in simulation of the three oil-water problems when the latter employed single relaxation factors. It took approximately 45 to 75 percent of the total computing time required by the best **SOR** technique namely **LSOR** with a unity relaxation factor being used.

In order to prevent growth of material balance errors, in their simulation of the oil-water problems with **SOR** they set the iteration parameter $\omega$ to unity without over-relaxation. A significant convergence rate with **PSOR** and **LSOR** was obtained.

**ADI** method converged in all the four problems. The **LSOR** technique was superior to **ADI** in simulation of the 100-grid block gas-oil cross section problem. The computing time requirement for **LSOR** using a single optimum relaxation factor and using a combination of factors were 83 and 76 percent, respectively, of the **ADI** computing time. **PSOR** simulation of this problem was unsuccessful. Their results are listed in Table 6.

In 1969, Weinstein et al. [6] developed **SIP** for the simultaneous solution of two or three coupled equations in two dimensions, such as those which arise in the

TABLE - 6 Summary of Comparative Results, ( Bjordammen et al., 1969).

| Problem No. | No of Time Steps | Method | Time (sec) | No. of Iterations | Time per Iteration (sec) |
|---|---|---|---|---|---|
| Original Model | | | | | |
| 1 | 60 | **ADI** | 53.6 | 492 | 0.1089 |
| | | **LSOR** ($\omega$=1.0) | 69.9 | 806 | 0.0866 |
| | | **PSOR** ($\omega$=1.0) | 101.5 | 1352 | 0.0752 |
| 2 | 46 | **ADI** | 308.8 | 530 | 0.5827 |
| | | **LSOR** ($\omega$=1.0) | 500.4 | 1067 | 0.4689 |
| | | **PSOR** ($\omega$=1.0) | 630.1 | 1523 | 0.4137 |
| 3 | 60 | **ADI** | 215.2 | 592 | 0.3635 |
| | | **LSOR** ($\omega$=1.0) | 349.9 | 1322 | 0.2646 |
| | | **PSOR** ($\omega$=1.0) | 405.5 | 1783 | 0.2274 |
| Modified Model | | | | | |
| 1 | 60 | **ADI** | 49.5 | 492 | 0.1001 |
| | | Improved **LSOR** | 46.6 | 539 | 0.0865 |
| | | **LSOR** ($\omega$=1.0) | 65.2 | 806 | 0.0809 |
| | | Improved **PSOR** | 61.1 | 827 | 0.0739 |
| | | **PSOR** ($\omega$=1.0) | 94.6 | 1352 | 0.0700 |
| 4 | 49 | **ADI** (4 cycle) | 72.6 | 831 | 0.0874 |
| | | **LSOR** ($\omega$=1.65) | 60.5 | 829 | 0.0730 |
| | | Improved **LSOR** | 55.0 | 741 | 0.0742 |
| | | **PSOR** | ---------- unsuccessful ---------- | | |

simultaneous-solution approach to multiphase two-dimensional flow problems. They presented the **SIP** algorithm for two-dimensional problems, evaluated it by several test problems and compared it with the **ADIP** method. The results for some of their studies, a gas-oil incompressible cross-section, a dissolved gas drive cross-section, and a water-oil radial coning problem, are listed in Table 7. They also tested SIP on two-dimensional three-phase problems, and reported some of the results.

They concluded that **SIP** is a fast reliable numerical iterative procedure for solving multiphase two-dimensional reservoir problems. On the problems tested, **SIP** required significantly less computational effort than **ADIP**. As the maximum transmissibility ratio increases, the work ratio increases. For a 12x20 2-D, two-phase, radial coning test, **ADIP** achieved only sporadic convergence, whereas **SIP** had no notable difficulty. They also found that **SIP** is less subject to rounding errors than **ADIP**.

The iteration time of **SIP** was modestly larger than **ADIP**; however, **SIP** generally required fewer iterations to converge. They presented some equations for determining the sequential iteration parameters for **SIP**.

In 1970, Watts [7] proposed an iterative method that is suitable for anisotropic problems. The method, called **LSORC**, consists of a correction applied at each mesh point in a line coupled with the **LSOR** method. That is to follow each **LSOR** sweep with a column correction. He tested the method on a problem from Stone's study [2] which has been discussed before. It is the 2-D, 31x31 homogenous, square problem with $k_x = k_y$ - isotropic conductivity, and $k_x = 100\ k_y$ - anisotropic conductivity. He compared the method with **LSOR**, **SIP**, and **ADI**, using results reported by Stone [2]. The comparison was made on the basis of required computational work. The results of the four methods for the isotropic

TABLE 7 - Two-Dimensional, Two-Phase Problems, ( Weinstein et al., 1969).

| Problem | ρ max | Method | Parameter (Max **SIP** Min **ADIP**) | No. of Time Steps | No. of Iter. | Average Iter. / Time Step | Iteration Time (millisec/ Iteration/ Block) | Total Time (minutes) | Work Ratio |
|---|---|---|---|---|---|---|---|---|---|
| Dissolved Gas | 66 | **SIP** | 0.99942, 1 | 8 | 189 | 23.6 | 2.41 | 3.83 | |
| Drive Case I | | **ADIP** | .0000569, 0 | 8 | 247 | 30.9 | 2.26 | 4.69 | 1.22 |
| Dissolved Gas | 660 | **SIP** | .9999942, 1 | 8 | 193 | 24.2 | 2.40 | 3.83 | |
| Drive Case II | | **ADIP** | .000023 , 0 | | | | | | |
| Gas -Oil | 3600 | **SIP** | 0.999998 | 25 | 77 | 3.1 | 1.97 | 0.993 | |
| Incomp. | | **ADIP** | $0.203 \times 10^{-4}$ | 25 | 389 | 15.6 | 1.48 | 3.75 | 3.78 |
| Water-Oil | 705000 | **SIP** | 0.9995 | 9 | 86 | 9.6 | 1.9 | 0.65 | |
| Radial Coning | | **ADIP** | 0.001 | 9 | | | | | |

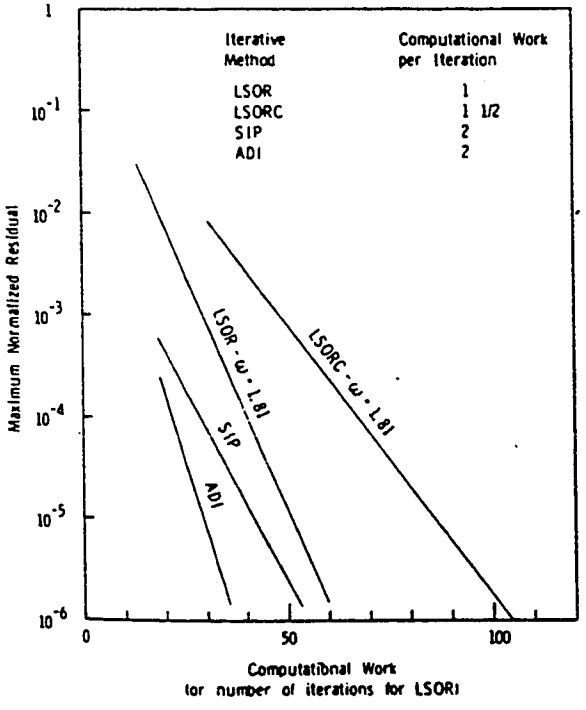



Figure 9 - Convergence Comparison, Homogeneous Square 31 x 31, $k_x = k_y$, (Watts, 1970)

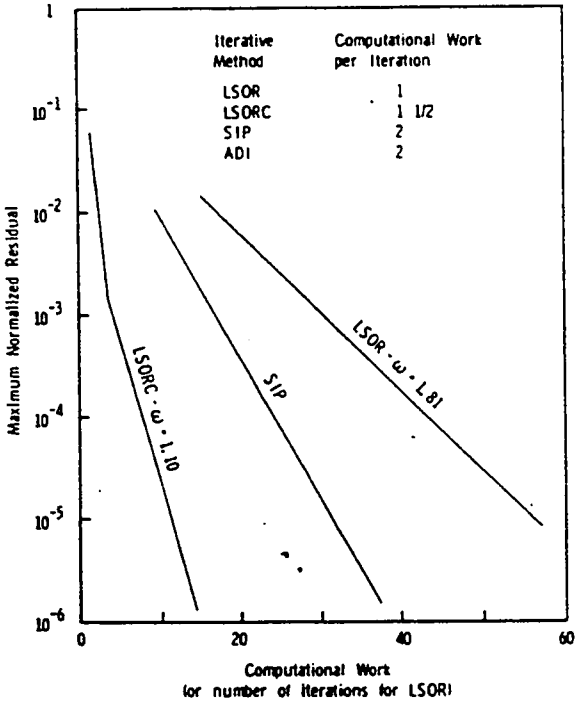



Figure 10 - Convergence Comparison, Heterogeneous Square 31 x 31, $k_x = 100\, k_y$, (Watts, 1970)

case are shown in Figure 9. As the Figure shows, **ADI** is the best, followed by **SIP**, then **LSOR** and finally **LSORC**. The results showed **LSORC** to be faster than the rest for the strongly anisotropic case, Figure 10, followed by **SIP**, and **LSOR**, while **ADI** was too slow.

In 1971, Traylor et al. [8] compared the computing efficiency of several methods : **LSOR**, semi iterative block Jacobi, iterative and non-iterative **ADI**, **SIP**, semi-iterative **LSSOR** (symmetric **LSOR**), and line Saul'Ev. A heterogeneous areal reservoir with square boundaries, and a vertical cross-section problem were used. The techniques were used to solve implicitly for pressures and explicitly for saturations. They investigated a new basis, the water-volume error, as a criterion to terminate the iterative methods. It involves the comparison of two predicted values of water saturation at each mesh point. They concluded that it is an acceptable method to determine the **L2** residual norm level required for a desired quality.

The noniterative methods such as Line Saul'Ev required extremely small time-step sizes to produce a small water-volume error, and thus the solution of reservoir problems with those methods is impractical. Table 8 shows the results of their comparison of the iterative methods. **LSOR** , **SIP** and **BJ (SI)** required small numbers of equivalent **LSOR** iterations, while **ADI** and **LSSOR (SI)** required larger numbers of iterations. Figures 11 and 12 present the average convergence of the five iterative techniques for the areal and cross section problems respectively.

They concluded that **LSOR** is a fast, efficient method of solving all the test problems, and that it has the advantage that good parameters for speeding its convergence may be estimated from the residual norms. **SIP** converged on all problems and proved to be effective. Its convergence rate was very sensitive to the value of the maximum iteration parameter when it is near unity. **BJ(SI)** also

TABLE 8 - Comparison of the Number of Iterations and Time Required, Areal Problem ( Traylor et al., 1971 )

| Iterative Method | cpu Time (seconds) | cpu sec / Time Step / Block | Total Iterations | Equivalent **LSOR** Iterations |
|---|---|---|---|---|
| **ADI** | 189.9 | .0165 | 393 | 786 |
| **LSOR** | 147.4 | .0128 | 556 | 556 |
| **LSSOR (SI)** | 163.6 | .0142 | 402 | 804 |
| **SIP** | 167.1 | .0145 | 241 | 482 |
| **BJ (SI)** | 154.6 | .0135 | 575 | 575 |

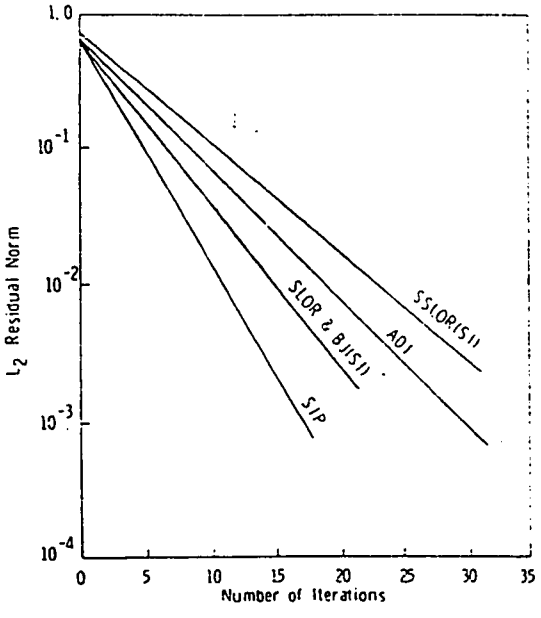



Figure 11 - Average Convergence on the Areal Problem
( Traylor et al., 1971)

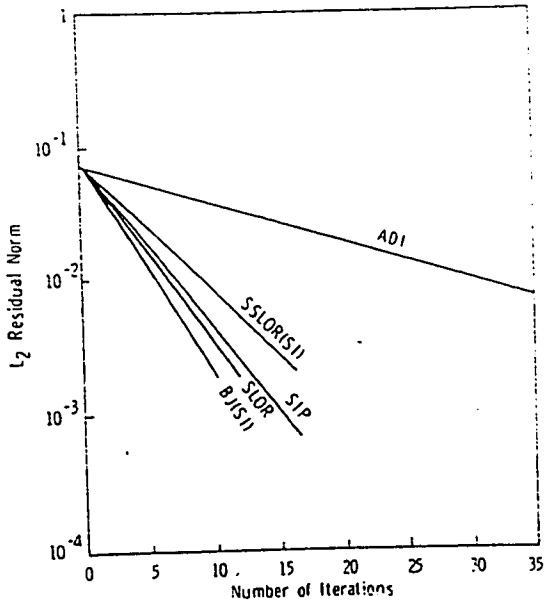



Figure 12 - Average Convergence on the Cross Section Problem
( Traylor et al., 1971)

proved to be a fast, efficient method of solving the test problems. **LSSOR** was sensitive to the heterogeneous nature of the areal problem and did not significantly reduce the cpu time required by **LSOR**. **ADI** was very sensitive to both the elongation and heterogeneities of the problems. The results for the cross-section problem are shown in Table 9.

In 1973, Price and Coats [9] developed several ordering schemes for Gaussian elimination that reduced computing times and storage requirements by factors as large as 6 and 3, respectively, relative to the standard ordering. The schemes tested were diagonal ordering, alternating point ordering, and the alternating diagonal ordering. Also, they compared their techniques against several commonly used iterative techniques, **LSOR**, **ADI** and **SIP**, using problems ranging from a simple homogeneous square to practical reservoir problems of typical heterogeneity and irregular geometry.

Table 10 shows the results obtained for an 18 x 12 x 6 reservoir problem. It shows the most efficient of the ordering schemes they developed was the alternating diagonal which they called **D4.** It was even faster than the iterative schemes. They pointed out the reason for the superiority of **D4** over the other techniques was a result from the missing grid blocks which were introduced into the problem. They also found out that for the iterative schemes such as **LSOR** the number of iterations increased as the time-step size was increased, Table 11.

Their results showed that the **D4** ordering has reduced the computing expense by a factor as large as 5.8 relative to the standard ordering. It also proved to be very competitive with some of the iterative methods, namely **SIP** and **LSOR** for full systems with nominal band widths up to 38.

In 1980, Al-Marhoun [10] proposed a new approach for ordering the system of grids and developed a direct solution algorithm. The modified ordering

TABLE 9 - Comparison of the Number of Iterations and Time Required, Cross Section Problem, ( Traylor et al., 1971 )

| Iterative Method | cpu Time (seconds) | cpu sec / Time Step / Block | Total Iterations | Equivalent **LSOR** Iterations |
|---|---|---|---|---|
| **ADI** | 271.9 | .0251 | 702 | 1404 |
| **LSOR** | 109.5 | .0101 | 331 | 331 |
| **LSSOR (SI)** | 109.4 | .0100 | 213 | 426 |
| **SIP** | 149.7 | .0138 | 206 | 412 |
| **BJ (SI)** | 117.6 | .0108 | 289 | 289 |

TABLE 10 - Comparison of Schemes, (Coats et al., 1973 ).

| | | Pressure | | | | Closure Criteria | |
|---|---|---|---|---|---|---|---|
| Method | $P_{avg}$ | Injection Well | Production Well | CDC6600 Computing Time (seconds) | Number of Iterations | $C_1$ | $C_2$ |
| **ADI** | -0.042 | 33.2996 | -42.6484 | 15.555 | 56 | 0.00171 | 0.234 |
| **LSOR** | 0.0058 | 33.3367 | -42.6258 | 13.55 | 107 | -0.00994 | 0.00994 |
| **SIP** | 2.53 | 34.8629 | -38.9796 | 9.43 | 40 | -0.102 | 0.251 |
| **D4** | 0. | 33.3284 | -42.6292 | 7.97 | | 0. | 0. |

TABLE 11 - **LSOR** Performance, (Coats et al., 1973 ).

| Time Step Number | Time-Step Size (days) | Time (days) | Optimum ω for **LSOR** | Number of **LSOR** Iterations | Maximum Change in Grid-Point Pressure over Step |
|---|---|---|---|---|---|
| 1 | 10 | 10 | 1.8357 | 49 | -92.9 |
| 2 | 23 | 33 | 1.8825 | 73 | -61.2 |
| 3 | 23 | 56 | 1.8825 | 63 | -32.5 |
| 4 | 23 | 79 | 1.8830 | 58 | -18.3 |
| 5 | 12.3 | 91.3 | 1.8494 | 40 | -6.9 |
| 6 | 91.25 | 182.5 | 1.9359 | 101 | -19.9 |
| 7 | 91.25 | 273.7 | 1.9359 | 123 | -68.6 |
| 8 | 91.25 | 365 | 1.9358 | 105 | -6.4 |

was to enclose the model areal grid into another system of grids and then apply an alternating diagonal scheme on the new grid system ( see Figure 30 ). This ordering, called restricted alternating diagonal ( **RAD** ) ordering, resulted into forming perfect secondary diagonals in the coefficient matrix. Thus it enabled applying the band matrix scheme to solve the simulation equations; therefore, a new solution procedure called **RAD** was developed.

The new approach was tested against **D4** coupled with the direct method of matrix de-composition, and the band-matrix algorithm using standard ordering called **STBAND**, in simulating a 345 grid heterogeneous system. The new technique proved to be substantially faster than both **STBAND** and **D4** schemes and also required less computer storage space. In a 600 days-simulation of a 345 cell system, **RAD** procedure required 65.5 percent of the computer time required for **STBAND** to perform the same simulation, Figure 13.

The new technique was also compared to several iterative methods namely **PSOR, LSOR, ADI,** and **SIP** [22]. The results are shown in Figure 14 and 15 for the homogenous and the heterogeneous cases, respectively. Among the iterative techniques that were tested, **PSOR** was the fastest. The **ADI** is slightly better for the homogenous case while **SIP** is considerably better for the heterogeneous case.

It was concluded that iterative methods required less computer storage, and were faster than the direct methods for the 345 cell model simulated; however, direct methods were more reliable than iterative ones and did not need the search for optimum iteration parameters, which was a serious problem in iterative methods especially for **SIP**.

In 1981, Watts [11] presented another technique using preconditioned conjugate gradients for solving the reservoir simulation pressure equation. He used a truncated direct elimination procedure to construct an

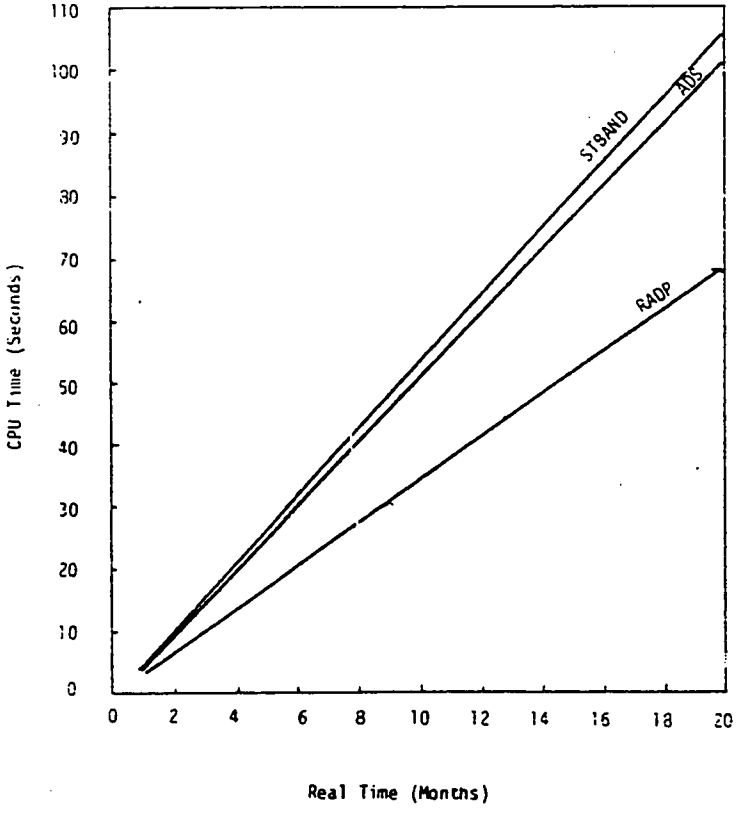



Figure 13 - cpu Time vs Real Time for 345 Cell Heterogeneous Case, ( Al-Marhoun, 1980)

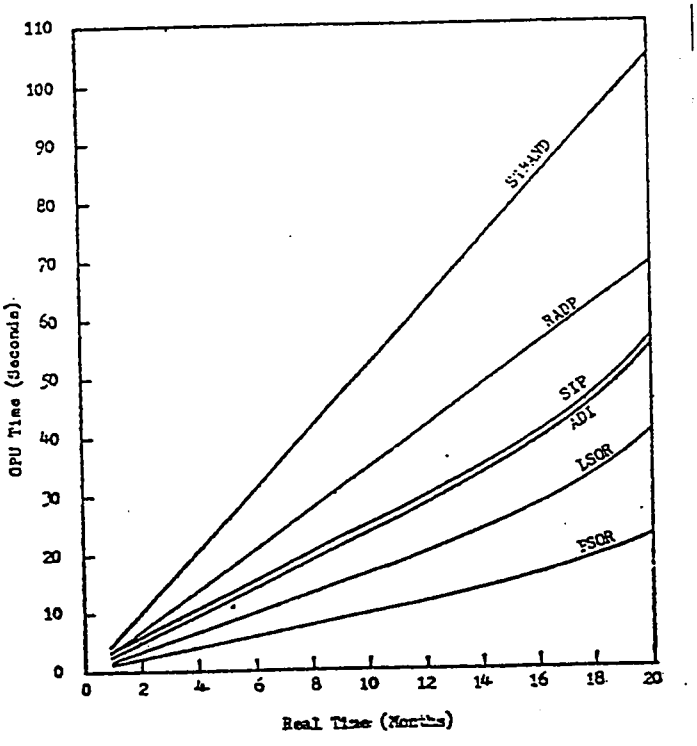



Figure 14 - cpu Time vs Real Time for 345 Cell Homogeneous Case, ( Al-Marhoun, 1978)

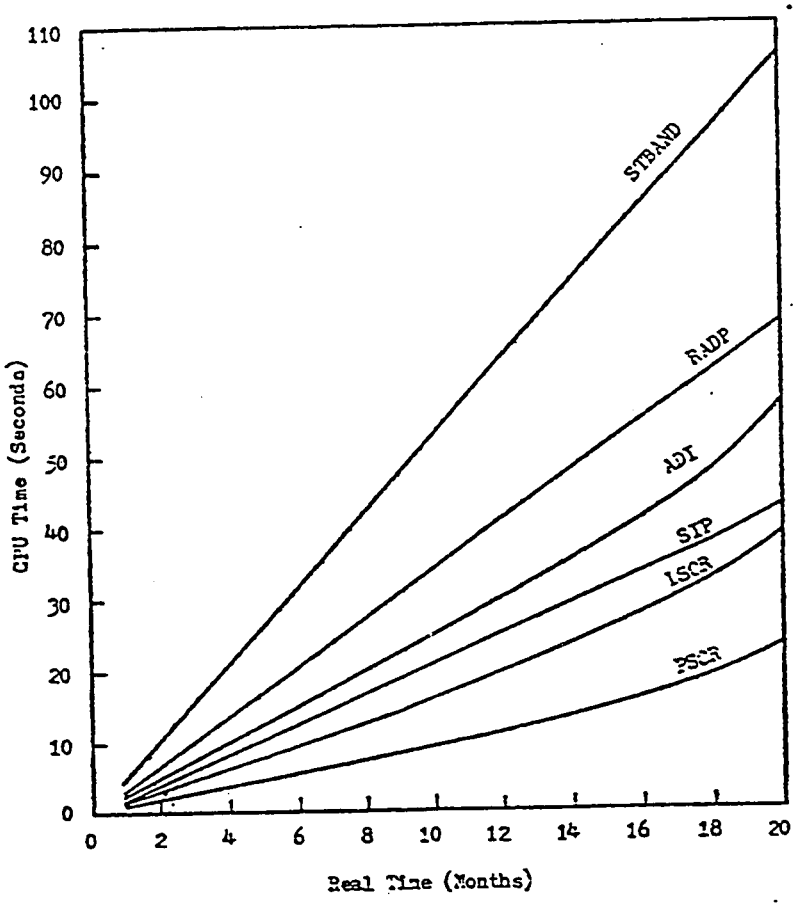



Figure 15 - cpu Time vs Real Time for 345 Cell Heterogeneous Case, ( Al-Marhoun, 1978)

TABLE 12 - Simplified Problem Test Results - Iterations Required for $10^{-6}$ Reduction in the Maximum Norm of the Residual ( Watts, 1981)

| Case | Dimensions | Iterations Required | | | Work Required by New Method - Equivalent SIP Iterations |
|---|---|---|---|---|---|
| | | New Method | SIP | LSORC | |
| 1 | 31x31 | 10 | 28 | 70 | 19 |
| 2 | 31x31 | 9 | 19 | 11 | 17.5 |
| 3 | 31x31 | 11 | 39 | - | 20.5 |
| 4 | 21x21x21 | 14 | 31 | - | 45 |
| 5 | 21x21x21 | 20 | 36 | - | 60 |
| 6 | 21x21x21 | 37 | 38 | - | 102.5 |
| 7 | 21x21x5 | 15 | 46 | 11 | 47.5 |
| 8 | 21x21x5 | 16 | 64 | 72 | 50 |

approximate inverse of a symmetric approximation to the coefficient matrix.

A comparison of the new method to **SIP** and corrected line successive overrelaxtion (**LSORC**) was performed in solution of several reservoir problems in both two and three dimensions. This new method required fewer iterations than **SIP** in all of the eight cases he tested, Table 12. It required less work than **SIP** in two dimensions but not in three. The new method was also tested againt **SIP** and **LSORC** in solving four practical field cases, which were a 10x20 coning problem, a 40x19 cross-section, a small 24 x 7 x 4 3-D problem, and a large 22 x 18 x 16 3-D problem. Table 13 summarizes the results of those four cases. The new method required fewer iterations than **SIP** in each case.

Watts concluded that the new method is quite reliable and its use seldom requires an estimation of any iteration parameter which gives it a significant advantage over **SIP**. It is generally faster than **SIP** in 2-D problems. However, in 3-D problems **SIP** is faster when it works well.

In 1983, Appleyard et al. [12] presented another iterative technique which comprises a novel preconditioning for the Conjugate Gradient and Orthomin iteration procedures. The new method, Nested Factorization, constructs block lower and upper factors using a procedure which adds one dimension at a time to the preconditioning matrix. Five versions of the new method were developed. They were called **NF1**, **NF2**, **NF3**, **NF4**, and **NF5**. They tested all of those versions along with **SIP** and some conjugate gradient methods, that used an incomplete Cholesky factorization for preconditioning, in solving a series of 2-D and 3-D problems. In all of the problems tested, all of the above versions of the new method worked well, but **NF1** was to be preferred because of its relative simplicity. They mentioned two significant advantages of the method which were : relatively modest storage requirements, and the low initialization cost.

TABLE 13 - Representative Field Data Test Results, ( Watts, 1981 ).

| | | | Average Iterations /Time Step | | | Average cpu Time/ Time Step ( seconds ) | | |
|---|---|---|---|---|---|---|---|---|
| CASE | Dimensions | Time Steps | New Method | SIP | LSORC | New Method | SIP | LSORC |
| 9 | 20x10 | 9 | 5.0 | 13 | - | 0.3 | 0.5 | - |
| 10 | 40x19 | 38 | 3.9 | 8.9 | - | 1.0 | 1.0 | - |
| 11 | 24x7x4 | 17 | 1.9 | 23 | 3 | 0.9 | 2.4 | 1.3 |
| 12 | 22x18x16 | 8 | 9.0 | 14 | - | 30 | 17.4 | - |

TABLE 14 - Comparison of Nested Factorization with Other Methods for Two-Dimensional Test Problems, ( Appleyard et al., 1983 ).

Computational Work ( **SIP** Iterations )

| Case Number | **SIP** | **ICCG0** | **ICCG3** | **NF1** | **NF2** | **NF3** | **NF4** | **NF5** |
|---|---|---|---|---|---|---|---|---|
| 1 | 28 | 30 | 19 | 12 | 11 | 10 | 9 | 10 |
| 2 | 20 | 24 | 27 | 6 | 6 | 6 | 13 | 5 |
| 3 | 38 | 33 | 21 | 12 | 10 | 10 | 13 | 12 |
| 4 | 28 | 29 | 19 | 13 | 11 | 10 | 9 | 12 |
| 5 | >50 | 32 | 20 | 11 | 11 | 9 | 9 | 10 |
| 6 | 50 | 25 | 14 | 9 | 6 | 7 | 10 | 9 |

TABLE 15 - Comparison of Nested Factorization with **SIP** and **CGTD** for a Number of Three-Dimensional Test Cases, ( Appleyard et al., 1983 ).

Computational Work ( **SIP** Iterations )

| Case | Dimensions | SIP | CGTD | NF1 | NF2 |
|---|---|---|---|---|---|
| 7 | 21x21x21 | 31 | 45 | 20 | 20 |
| 8 | 21x21x21 | 36 | 60 | 11 | 12 |
| 9 | 21x21x21 | 38 | 102 | 16 | 15 |
| 10 | 21x21x5 | 46 | 47 | 11 | 12 |
| 11 | 21x21x5 | 64 | 50 | 16 | 16 |

They also tested the methods on the two-dimensional single phase problems described by Stone [2] , and Settari and Aziz [13]. The results showed this method to converge faster than the other methods as well, Table 14. They also tested the methods on a series of single phase test problems, and compared their results with the results reported by Watts [11]. Table 15. **NF1** compares well with the other methods, and appeared to show an even marked advantage over those methods in terms of computational work.

In 1987, Harouaka [14] developed a direct block sparse solution algorithm, tested it, and applied it to solve case one of the study done by Odeh [15]. He developed a fully implicit three-phase, three-dimensional black oil simulator strongly coupled with a multilayer completion well model. The performance of this solver was compared to three other solution techniques. Two direct techniques which were block-band and **Gband**. The other was the iterative technique block line successive over-relaxation ( **BSOR** ). The block sparse solver was tested with natural ordering and with **D4** ordering keeping the Jacobian constant once every other Newton-Raphson iteration.

It was concluded that the block sparse approach was superior to the other direct techniques from almost any perspective. Under similar convergence criteria, it was found that **BSOR** and the direct solution techniques did not converge to the same values. All direct solvers, on the other hand, converged exactly to the same value. It was also concluded that the accuracy of **BSOR** improves with smaller convergence criteria.

Harouaka also concluded that direct methods are more accurate than iterative methods, and the block sparce approach leads to a considerable reduction in overhead storage and probably in overall storage. Furthermore, keeping the Jacobian constant over one Newton-Raphson iteration results in a considerable

saving in computer time.

From the above literature review, an up-to-date comparative study of those computational solution methods is worthy. This thesis is intended to accomplish the objective of a complete up-to-date comparative study that includes the most commonly used methods. Those methods will be tested to solve a given two-dimensional, two-phase reservoir problem. A comparison will be made of their computing efficiencies in terms of the simulation times as well as the memory space requirements. Also an analysis of the different methods in terms of their performance and limitations will be made. The study will be performed utilizing an existing two-phase reservoir simulator. In the simulation, the different solution techniques will be used to solve for the pressures implicitly and for saturations explicitly. The methods that will be studied are as follows :

A - Direct methods :

1 - Gaussian Elimination using standard ordering,
( **STANDARD - GAUSS** ) .

2 - Gaussian Elimination using alternating diagonal ordering,
( **D4 - GAUSS** ).

3 - Matrix Decomposition using standard ordering, ( **STBAND** ).

4 - Restricted Alternating Diagonal Method, ( **RAD** ).

B - Iterative methods :

1 - Point Successive Over-Relaxation ( **PSOR** ).

2 - Line Successive Over-Relaxation ( **LSOR** ).

3 - Alternating Direction Implicit Procedure ( **ADIP** ).

4 - Strongly Implicit Procedure (**SIP**).

5 - Conjugate Gradient-Truncated Direct method ( **CGTD** ).

6 - Nested Factorization ( **NF** ).

# CHAPTER 3

# DIRECT METHODS

A variety of direct solution methods have been developed for solving the large matrix of equations which arise from the approximation of multi-dimensional partial differential equations. Direct methods are those in which the solution to the system of equations is obtained upon the completion of a fixed number of operations. Some examples of the direct methods to solve those sets of equations are :

1. Gaussian elimination.
2. Gauss-Jordan method.
3. Matrix decomposition.

Those direct processes are explained fully in textbooks on reservoir simulation [16]. The most popular direct methods used in reservoir simulation employ either the Gaussian elimination or the matrix decomposition processes in their solutions. These two processes are explained in the next sections.

## 3.1 Gaussian Elimination

This is a systematic technique which involves a two-step algorithm. In the initial pass the matrix is converted to an upper triangular matrix, the operation being carried out on the right-hand side vector of constants at the same time. In the final pass the solution is obtained by backward substitution for the unknowns [16]. The algorithm for this process is as follows:

Given:

$$\mathbf{A}\,\mathbf{P} = \mathbf{d} \tag{3.1}$$

where

**A** = The coefficients matrix

**P** = The pressure matrix

**d** = Right-hand side constants

Forward reduction:

$$c = \frac{a_{ik}}{a_{kk}}$$

$$a_{ij} = a_{ij} - c\,(a_{kj}) \tag{3.2}$$

where

$j = k, \ldots, N+1$

$i = k+1, \ldots, N$

$k = 1, 2, \ldots, N-1$

The right-hand column of constants is augmented to the original matrix to make it an $[N \times (N+1)]$ matrix. This allows the same operations to be carried out on the vector of constants.

Back substitution:

$$P_N = \frac{a_{N,N+1}}{a_{NN}} \tag{3.3}$$

$$P_i = - \frac{a_{i,N+1} - \sum_{j=i+1}^{N} a_{ij} P_j}{a_{ii}} \quad \text{for } i = N-1, N-2, \ldots, 1 \tag{3.4}$$

The computation time of this method is constant and is equal to

$$1/3\ N^3 + N^2 + O(N)$$

where N is the number of cells in the simulation grid.

## 3.2 L / U Decomposition

This process involves the transformation of a matrix into other matrices which are generally easier to operate on and then using these transformed matrices to obtain the solution. One method attributed to Crout [17] performs a decomposition of the matrix into a lower triangular and an upper triangular matrix. This decomposition is followed by a back substitution which computes the answer in two successive substitution steps. The process is as follows:

Given equation 3.1 above:

$$\mathbf{A\,P = d}$$

where

**A** = Coefficients matrix

**P** = Pressure matrix

then $$\mathbf{A = L\,U} \tag{3.5}$$

The matrix **A** generated for a 5 x 5 two-dimensional grid system is shown in Figure 16. The upper matrix **U** and the lower matrix **L** are shown in Figures 17 and 18, respectively. From these triangular matrices we can compute the solution vector **P** as follows:

From equation 3.5, since

$$\mathbf{A = L\,U}$$

Then by substitution into equation 3.1 we get

$$\mathbf{L\,U\,P = d} \tag{3.6}$$

If we call the product **UP** the vector **y**, then from equation 3.6 we get

$$\mathbf{L\,y = d} \tag{3.7}$$

and

$$\mathbf{U\ P = y} \tag{3.8}$$

Solving equations 3.7 and 3.8 successively for **y** and **d** respectively

$$\mathbf{y = L^{-1}\ d} \tag{3.9}$$

and

$$\mathbf{P = U^{-1}\ y} \tag{3.10}$$

The algorithm for this process is as follows. The elements of the lower matrix are $l_{ij}$, and $l_{ij} = 0$ if $i < j$, and the elements of the upper are $u_{ij}$ with $u_{ii} = 1$ if $i > j$.

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 6 | 7 | 8 | 9 | 10 |
| 3 | 11 | 12 | 13 | 14 | 15 |
| 4 | 16 | 17 | 18 | 19 | 20 |
| 5 | 21 | 22 | 23 | 24 | 25 |

Figure 16 - A 5 x 5 Two-Dimensional Grid

$$\begin{bmatrix} l_{11} & & & & & & \\ l_{21} & l_{22} & & & & & \\ l_{31} & l_{32} & l_{33} & & & & \\ l_{41} & l_{42} & l_{43} & l_{44} & & & \\ & & & & & & \\ \vdots & & & & & & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & & & l_{nn} \end{bmatrix}$$

Figure 17 - The Lower Matrix , n = 25

$$\begin{bmatrix} 1 & u_{12} & u_{22} & . & . & . & u_{1n} \\ & 1 & u_{23} & & & & u_{2n} \\ & & 1 & u_{34} & & & u_{3n} \\ & & & 1 & & & u_{4n} \\ & & & & & & . \\ . & & & & & & . \\ . & & & & & & . \\ . & & & & & & . \\ & & & . & . & . & 1 \end{bmatrix}$$

Figure 18 - The Upper Matrix, n = 25

Then

$$l_{im} = a_{im} - \sum_{k=1}^{m-1} l_{ik}\, u_{km} \quad \text{for} \quad i = m, m+1, \ldots, n \qquad (3.11)$$

$$m = 1, 2, \ldots, n$$

$$u_{mj} = \frac{1}{l_{mm}} \left( a_{mj} - \sum_{k=1}^{m-1} l_{mk}\, u_{kj} \right) \qquad (3.12)$$

$$\text{for} \quad j = m+1, m+2, \ldots, n$$

$$m = 1, 2, \ldots, n$$

Back substitution:

$$y_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik}\, y_k}{l_{ii}} \qquad \text{for} \quad i = 1, 2, \ldots, n \qquad (3.13)$$

and

$$P_i = y_i - \sum_{k=i+1}^{n} u_{ik}\, P_k \qquad \text{for} \quad i = n, n-1, \ldots, 1 \qquad (3.14)$$

The computation time of this method is also constant and is equal to

$$1/3\, N^3 + N^2 + O(N)$$

where N is the number of cells in the simulation grid.

The above direct approaches are the most efficient methods available for

solving small sets of equations but not for large sets. Direct solutions have been found more reliable, but for large size problems they normally require a larger storage space and more computer time than the iterative methods. Therefore, the choice of those solution technique will generally depend on the nature and the size of the reservoir being modeled.

## 3.3 Ordering Schemes

Different schemes for ordering the reservoir problem grids have appeared in the literature. The natural ( sometimes called standard ordering ) in reservoir simulation is to number the points of a three-dimensional grid first along the shortest direction, then in the next shortest direction, and finally in the longest direction. Figure 19 shows how this ordering is applied to a 5 x 6 2-D grid. The shape of the matrix resulting from such an ordering is shown in Figure 20. The work ( W ) and storage requirements for this ordering were found to be roughly

$$W = I\ J^3$$

$$S = I\ J^2$$

The work is defined as the number of multiplications and divisions necessary to solve for the unknowns without operating on zero elements. The storage is required only for nonzero elements.

The ordering of the grid system have been found to play a big role in the effectiveness of the direct methods. Different ordering of the grid system causes the coefficient matrix of the linear system of equations to take different forms.

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 6 | 7 | 8 | 9 | 10 |
| 3 | 11 | 12 | 13 | 14 | 15 |
| 4 | 16 | 17 | 18 | 19 | 20 |
| 5 | 21 | 22 | 23 | 24 | 25 |
| 6 | 26 | 27 | 28 | 29 | 30 |

Figure 19 - Standard Ordering Applied
to a 5 x 6 2-D Grid

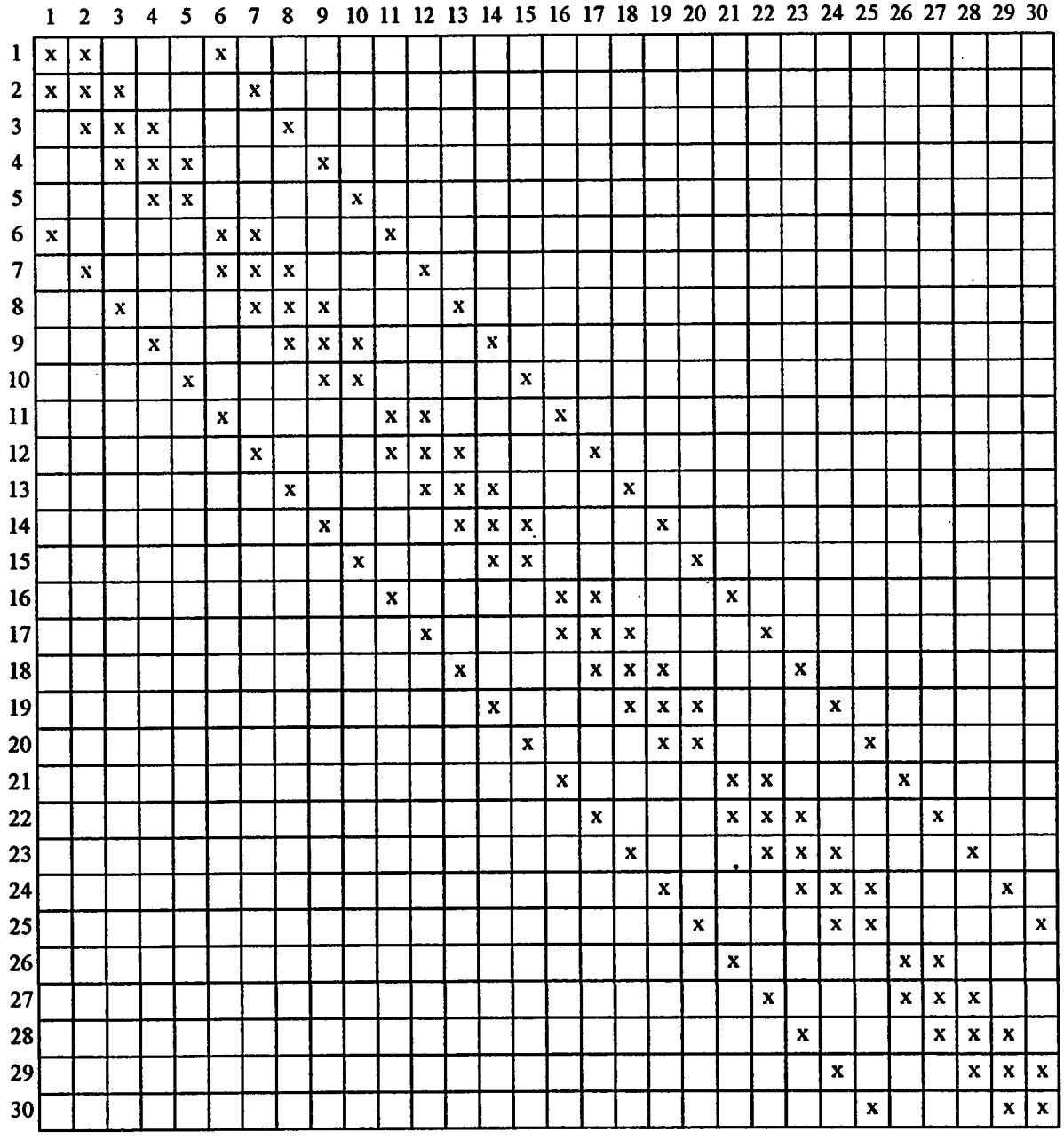


Figure 20 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 19, Using Standard Ordering

Price and Coats [9] presented several ordering schemes which are :

1. Diagonal ordering, D2
2. Alternating point ordering, A3
3. Alternating diagonal ordering, D4

A description of each scheme follows.

### 3.3.1 Diagonal Ordering , D2

This ordering numbers the grid along diagonals as shown in Figure 21. The matrix produced from such an ordering for the 5 by 6 grid system is shown in Figure 22. The cells are numbered consecutively starting with the shortest direction. This method groups the cells by diagonal count, which by inspection increases as we move from the upper left corner through the grid to the lower right corner. The band width is seen to increase with increasing i and then shrinks again. Therefore, it is important to select the shorter direction as the primary direction for this ordering. The work and storage requirements were found to be roughly

$$W = IJ^3 - \frac{J^4}{2}$$

$$S = IJ^2 - \frac{J^3}{3}$$

### 3.3.2 Alternating Point Ordering, A3

This ordering numbers the grid every other point, Figure 23. The matrix

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 2 | 4 | 7 | 11 |
| 2 | 3 | 5 | 8 | 12 | 16 |
| 3 | 6 | 9 | 13 | 17 | 21 |
| 4 | 10 | 14 | 18 | 22 | 25 |
| 5 | 15 | 19 | 23 | 26 | 28 |
| 6 | 20 | 24 | 27 | 29 | 30 |

Figure 21 - Diagonal Ordering Applied to the 5 x 6 2-D Grid

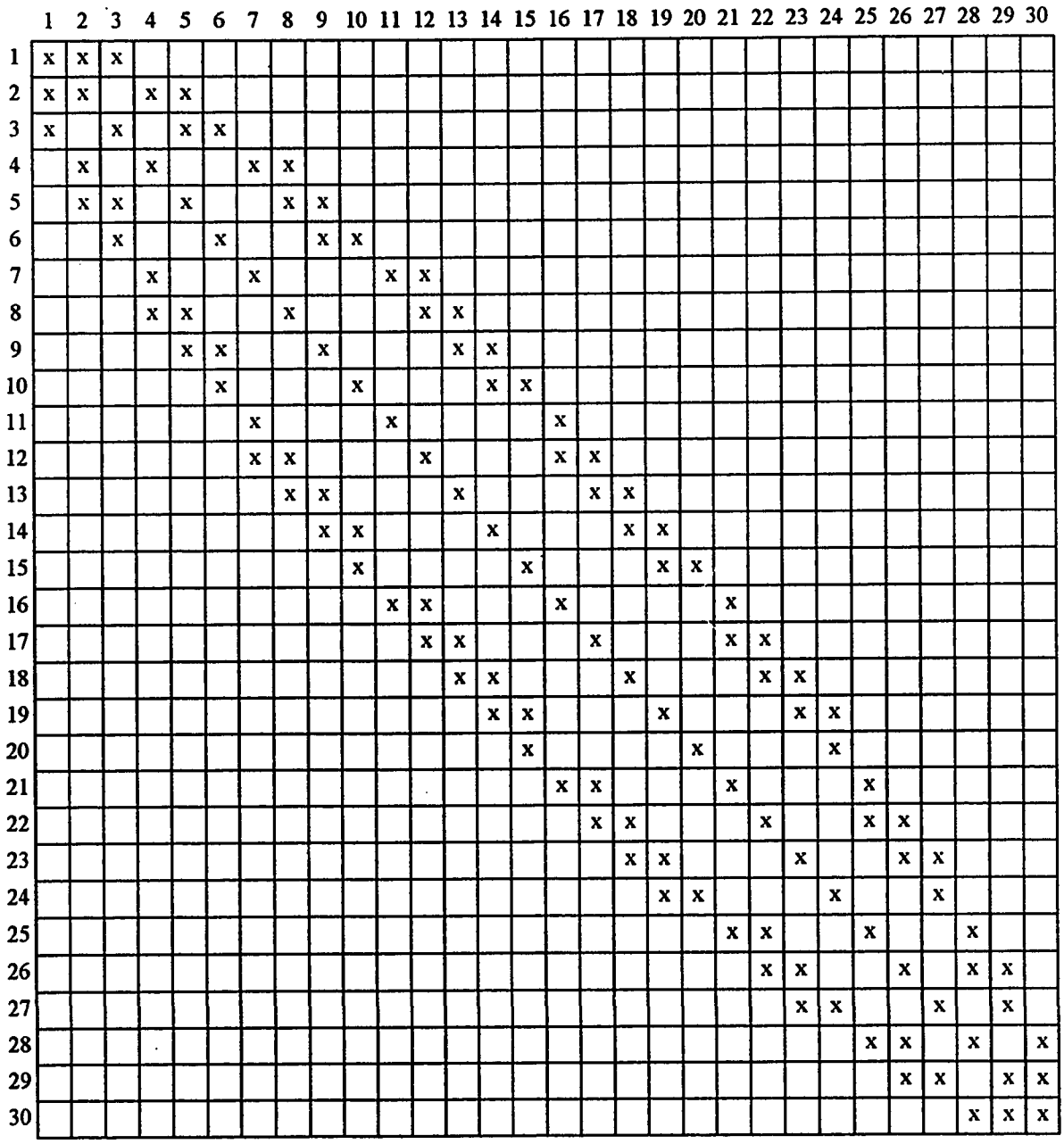


Figure 22 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 21, Using Diagonal Ordering.

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 16 | 2 | 17 | 3 |
| 2 | 18 | 4 | 19 | 5 | 20 |
| 3 | 6 | 21 | 7 | 22 | 8 |
| 4 | 23 | 9 | 24 | 10 | 25 |
| 5 | 11 | 26 | 12 | 27 | 13 |
| 6 | 28 | 14 | 29 | 15 | 30 |

Figure 23 - Alternating Point Ordering Applied to a 5 x 6 2-D Grid

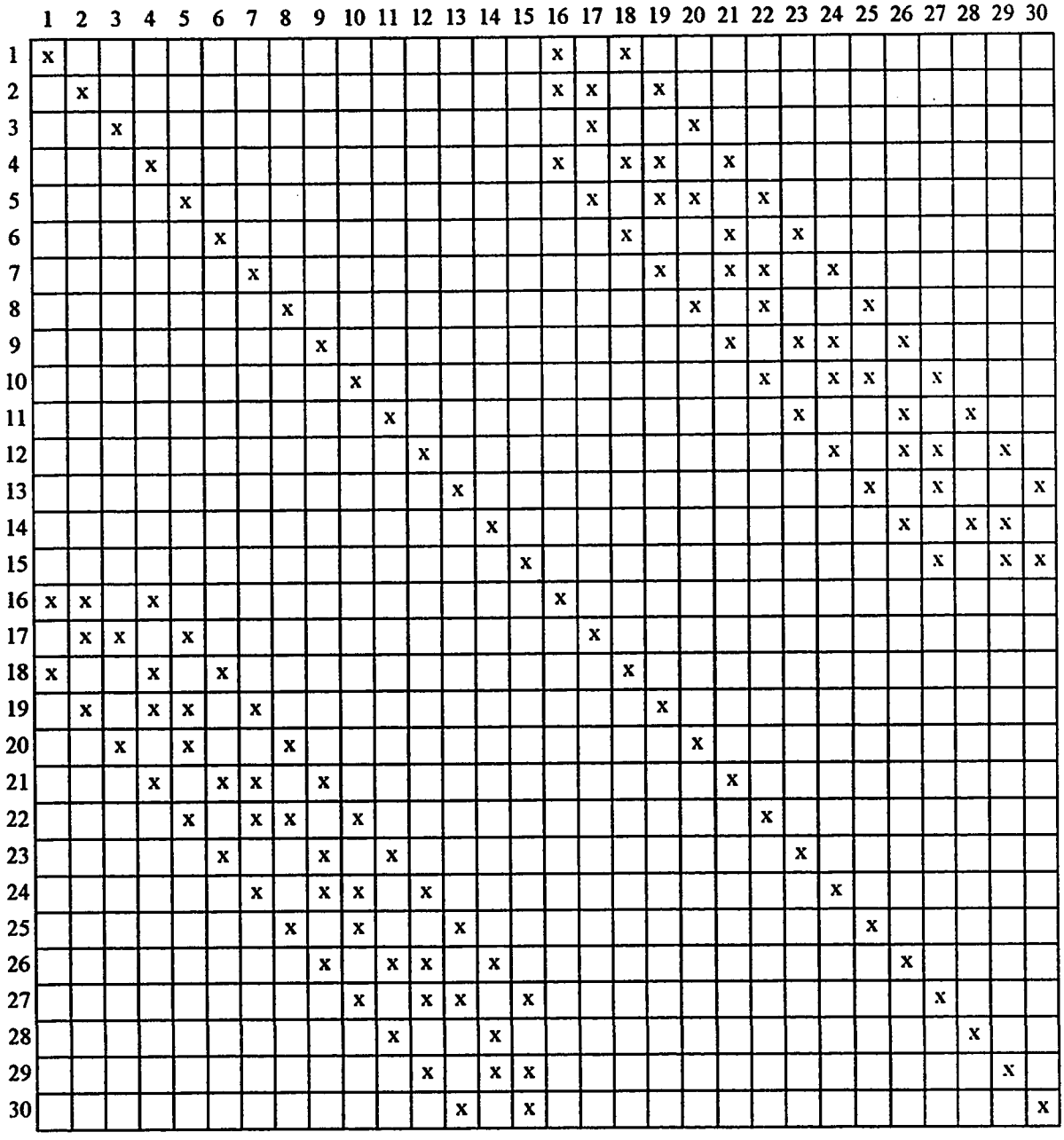


Figure 24 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 23, Using Alternating Point Ordering.

produced from such an ordering for the 5 x 6 grid system is shown in Figure 24. The cells are numbered alternately starting with the shortest direction. By inspection of the matrix produced from this ordering, the band width is seen to be of a constant width. The work and storage requirements are roughly

$$W = \frac{IJ^3}{2}$$

$$S = \frac{IJ^2}{2}$$

Thus for a square I = J grid, this ordering requires one-half the work and the storage of the standard ordering.

### 3.3.3 Alternating Diagonal Ordering, D4

This scheme orders the grid points on alternating diagonals, Figure 25. Thus, it is a combination of the alternating point and diagonal orderings. It was found to produce the greatest reduction in work. The matrix produced from such an ordering for the 5 x 6 grid system is shown in Figure 26. The cells are numbered on alternating diagonals starting with the shortest direction. By inspection of the matrix produced from this ordering, the band width is variable and it does not form perfect secondary diagonals. The work and storage requirements for this ordering are roughly

$$W = \frac{IJ^3}{2} - \frac{J^4}{4}$$

$$S = \frac{IJ^2}{2} - \frac{J^3}{6}$$

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 16 | 2 | 18 | 5 |
| 2 | 17 | 3 | 19 | 6 | 22 |
| 3 | 4 | 20 | 7 | 23 | 10 |
| 4 | 21 | 8 | 24 | 11 | 27 |
| 5 | 9 | 25 | 12 | 28 | 14 |
| 6 | 26 | 13 | 29 | 15 | 30 |

Figure 25 - Alternating Diagonal Ordering Applied to a 5 x 6 2-D Grid

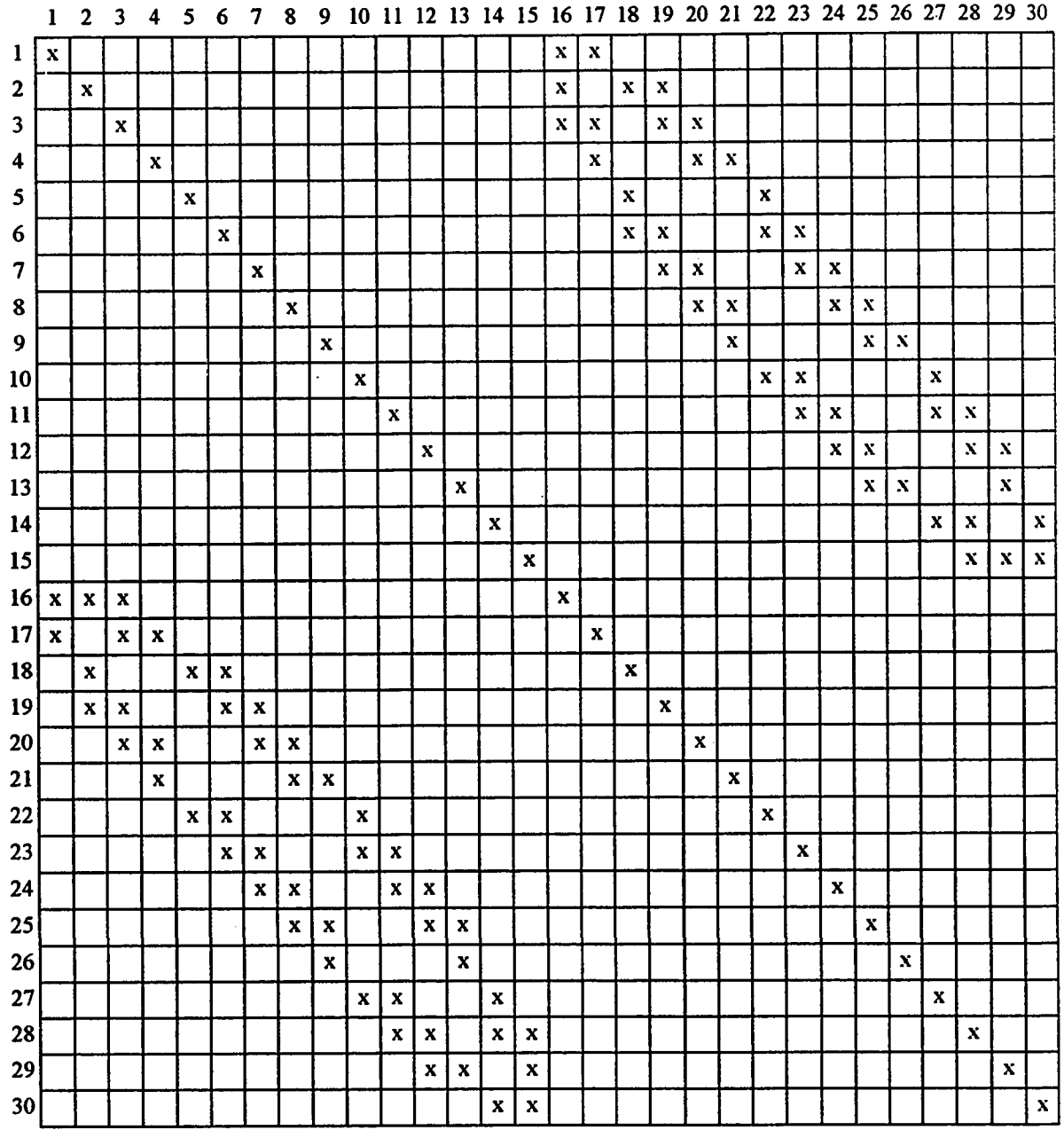


Figure 26 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 25, Using Alternating Diagonal Ordering.

Thus for a square I = J grid, this ordering requires less work and storage than the above two orderings.

### 3.3.4 Alternating Line Ordering

There are other varieties of orderings. One other ordering is the alternating line ordering. This scheme orders the grid points on alternating lines, Figure 27, starting with the shortest row or column. The matrix produced from such an ordering for the 5 x 6 grid system is shown in Figure 28. The cells are numbered on alternating lines consecutively. The matrix produced from this ordering is characterized with a fixed band width. The work and storage requirements for this scheme are:

$$W = \frac{IJ^3}{2}$$

$$S = \frac{IJ^2}{2} - \frac{J^3}{6}$$

## 3.4 Sparse Matrix Techniques

The matrices arising in reservoir simulation problems are such that most of their elements are zeros. These matrices are known as sparse matrices. There are some techniques that take advantage of the sparse character of the matrix. Certain steps are involved in the application of those sparce matrix techniques.

One method is the generate-and-solve algorithm developed by Woo, et al. [17]. Basically the idea of this process is to generate a code and then solve the

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 16 | 17 | 18 | 19 | 20 |
| 3 | 6 | 7 | 8 | 9 | 10 |
| 4 | 21 | 22 | 23 | 24 | 25 |
| 5 | 11 | 12 | 13 | 14 | 15 |
| 6 | 26 | 27 | 28 | 29 | 30 |

Figure 27 - Alternating Line Ordering Applied to a 5 x 6 2-D Grid

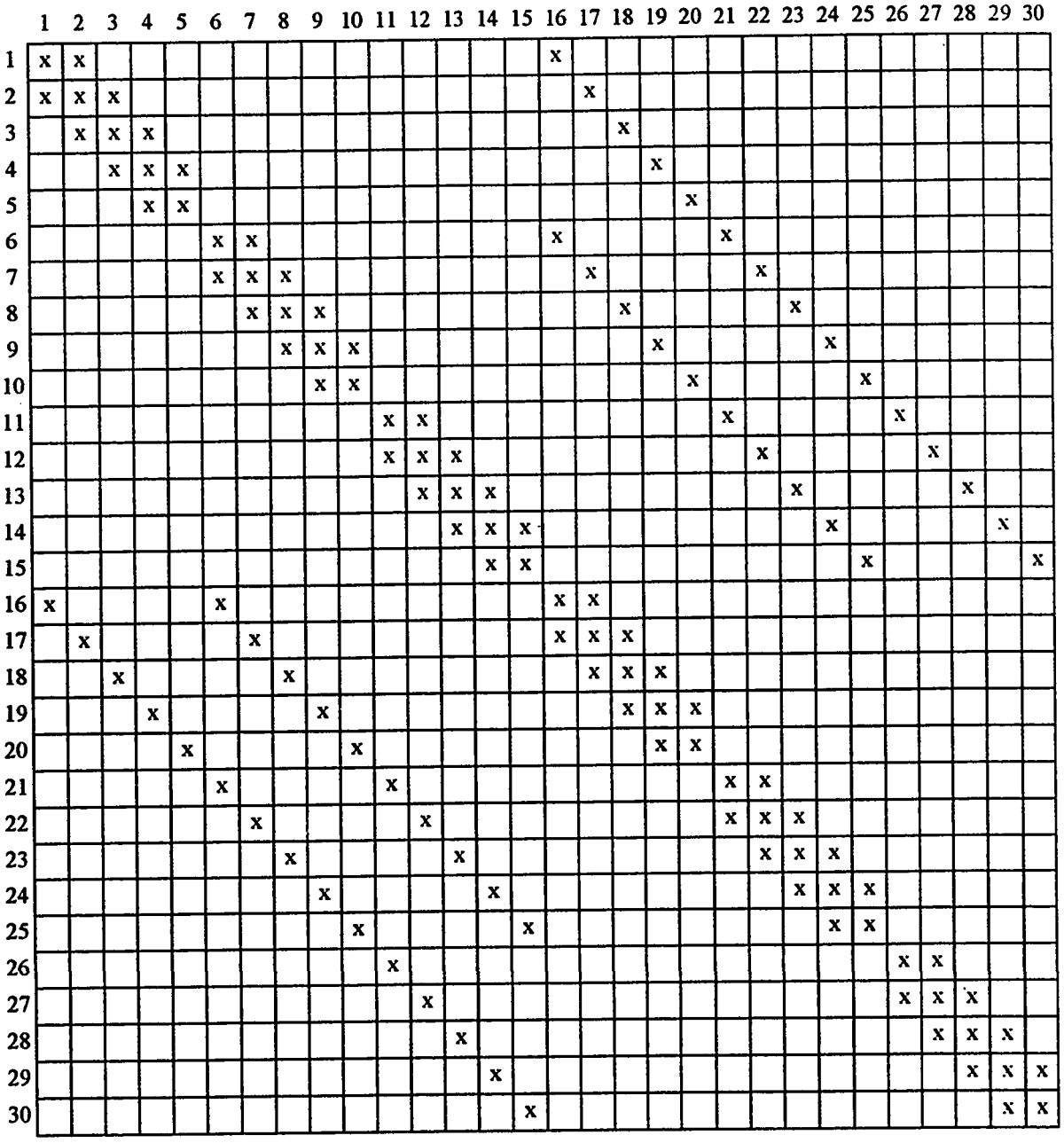


Figure 28 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 27, Using Alternating Line Ordering.

problem. The basis of the method as described by Crichlow [16] is as follows :

1. In a given matrix there is a fixed number of nontrivial operations rquired to solve the system by some given direct scheme.
2. The solution process produces some nonzero elements which will always be present at a given ( i , j ) location in the matrix.
3. If the location of all nonzero quantities is fixed, a priori both in the matrix and in its solution, then a straightforward, nonbranching, nonlooping computer code can be developed to solve this system very efficiently.

This process however is found to require large storage and overhead work to store the nonlooping computer code.

Other sparse matrix techniques are the band matrix techniques. Because the direct methods usually require large computer storage space and long computation times, special techniques that utilize the band matrix or sparsity structures are used to solve the pressure system in simulation. The treatment of zeros within the bands as non-zeros is called band matrix technique. One popular method which applies this technique is the general band solver. Al-Marhoun [10] applied one such band matrix algorithm on standard ordering. He called it **STBAND** and his description is as follows.

Refer to Figure 29. All the elements between the E-vector and the F-vector are treated as nonzeros.

By letting

$$G_{n,j} = e_j \tag{3.15}$$

$$G_{1,j} = a_j \tag{3.16}$$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | b | c | | | | f | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | a | b | c | | | | f | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | a | b | c | | | | f | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | a | b | c | | | | f | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | a | b | | | | | f | | | | | | | | | | | | | | | | | | | | |
| 6 | e | | | | | b | c | | | | f | | | | | | | | | | | | | | | | | | | |
| 7 | | e | | | | a | b | c | | | | f | | | | | | | | | | | | | | | | | | |
| 8 | | | e | | | | a | b | c | | | | f | | | | | | | | | | | | | | | | | |
| 9 | | | | e | | | | a | b | c | | | | f | | | | | | | | | | | | | | | | |
| 10 | | | | | e | | | | a | b | | | | | f | | | | | | | | | | | | | | | |
| 11 | | | | | | e | | | | | b | c | | | | f | | | | | | | | | | | | | | |
| 12 | | | | | | | e | | | | a | b | c | | | | f | | | | | | | | | | | | | |
| 13 | | | | | | | | e | | | | a | b | c | | | | f | | | | | | | | | | | | |
| 14 | | | | | | | | | e | | | | a | b | c | | | | f | | | | | | | | | | | |
| 15 | | | | | | | | | | e | | | | a | b | | | | | f | | | | | | | | | | |
| 16 | | | | | | | | | | | e | | | | | b | c | | | | f | | | | | | | | | |
| 17 | | | | | | | | | | | | e | | | | a | b | c | | | | f | | | | | | | | |
| 18 | | | | | | | | | | | | | e | | | | a | b | c | | | | f | | | | | | | |
| 19 | | | | | | | | | | | | | | e | | | | a | b | c | | | | f | | | | | | |
| 20 | | | | | | | | | | | | | | | e | | | | a | b | | | | | f | | | | | |
| 21 | | | | | | | | | | | | | | | | e | | | | | b | c | | | | f | | | | |
| 22 | | | | | | | | | | | | | | | | | e | | | | a | b | c | | | | f | | | |
| 23 | | | | | | | | | | | | | | | | | | e | | | | a | b | c | | | | f | | |
| 24 | | | | | | | | | | | | | | | | | | | e | | | | a | b | c | | | | f | |
| 25 | | | | | | | | | | | | | | | | | | | | e | | | | a | b | | | | | f |
| 26 | | | | | | | | | | | | | | | | | | | | | e | | | | | b | c | | | |
| 27 | | | | | | | | | | | | | | | | | | | | | | e | | | | a | b | c | | |
| 28 | | | | | | | | | | | | | | | | | | | | | | | e | | | | a | b | c | |
| 29 | | | | | | | | | | | | | | | | | | | | | | | | e | | | | a | b | c |
| 30 | | | | | | | | | | | | | | | | | | | | | | | | | e | | | | a | b |

Figure 29 -30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 19, Using Standard Ordering.

$$H_{1,j} = c_j \tag{3.17}$$

$$H_{n,j} = f_j \tag{3.18}$$

the general equation of **STBAND** would be

$$G_{1,j} P_{j-n} + G_{2,j} P_{j-n+1} + \ldots + G_{n,j} P_{j-1} + b_j P_j + H_{n,j} P_{j+1} + H_{n-1} P_{j+2} + \ldots + H_{1,j} P_{j+n} = d_j \tag{3.19}$$

$$1 \leq N,\ N \geq n$$

The algorithm of the method is as follows:

$$\alpha_{i,j} = G_{i,j} = 0 \qquad i \leq j \tag{3.20}$$

$$H_{i,j} = 0 \qquad \text{for} \quad i \geq N+1-j \tag{3.21}$$

Forward solution ( $j = 1, 2, \ldots, N$ )

$$\alpha_{i,j} = G_{i,j} - \sum_{k=i+1}^{k=\eta} (\alpha_{k,j}\, y_{k-i,j-k}) \qquad i = \eta, \eta-1, \ldots, 1 \tag{3.22}$$

$$\beta_j = b_j - \sum_{k=1}^{k=\eta} (\alpha_{k,j}\, y_{k,j-k}) \tag{3.23}$$

$$y_{i,j} = \frac{1}{\beta_j} \{ H_{i,j} - \sum_{k=i+1}^{k=\eta} (\alpha_{k-i,j}\, y_{k,j-k+i}) \}, \tag{3.24}$$

$$i = 1, 2, \ldots, \eta$$

$$\gamma_j = \frac{1}{\beta_j} \{ d_j - \sum_{k=1}^{k=\eta} (\alpha_{k,j} \gamma_{j-k}) \}, \quad i= 1,2, \ldots, \eta \quad (3.25)$$

Backward Solution ( j = N, N - 1 , ... , 1 )

$$P_j = \gamma_j - \sum_{k=1}^{k=\eta} (y_{k,j} P_{j+k}) \quad (3.26)$$

Various band matrix algorithms can also be developed using either of the direct processes, *Matrix De-composition* or *Gaussian Elimination* techniques, coupled with various ordering techniques.

## 3.5 Restricted Alternating Diagonal Method

Another direct method is that of Al-Marhoun [10]. He proposed another ordering technique. His technique was to re-arrange each type of element along different secondary diagonals in the coefficient matrix. He called it restricted alternating diagonal ordering (**RAD**).

As previously mentioned, the alternating diagonal ordering ( **D4** ) has been so far found to be the optimal ordering in petroleum reservoir simulation. A drawback to this scheme is that the elements do not form perfect secondary diagonals as shown in Figure 26. Using the band algorithm with this ordering will require us to treat the zeros within the band as nonzeros. This will of course require a larger storage space and a more computational labor.

With a new approach, Al-Marhoun modified the cell ordering by adding

more cells and forming another grid that encloses the original grid as is shown in Figure 30. Then applying the alternating diagonal ordering on the new grid in such a way as to produce perfect secondary diagonals as is shown in the resulting matrix in Figures 31 to 34.

With this arrangement, there will be no new elements produced during factorization except in the lower right corner of the composite matrix, Figure 35. The locations of the original and the new elements were known before hand, and therefore he could develope a general solution procedure to solve the system of equations. This procedure eliminated the overhead calculations and extra storage needed for the zero elements in the band matrix algorithm.

## LU Factorization Applied to RAD

The process of **LU** factorization as applied to this ordering is as follows :

1. The standard ordering is reordered to **RAD** scheme as shown in Figure 30. The actual system grid is embedded in the smallest diagonal grid to generate the desired ordering. However, only the cells in the actual grid are involved in the computation.
2. The coefficient matrix is transformed into the **RAD** scheme.
3. After solution, the pressure vector obtained is reordered into standard ordering to go along with the rest of the simulation package.

The computation of the composite matrix **LU** is as follows:

For $j = 1, 2, \ldots, N_u$

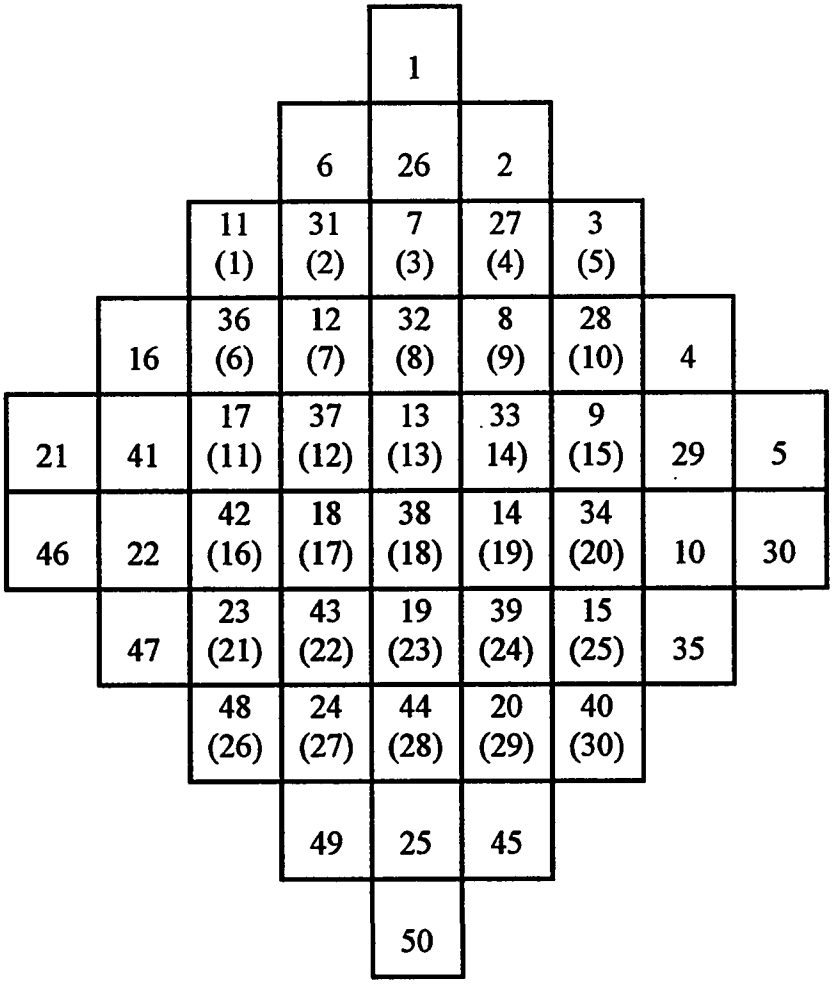



Figure 30 - Restricted Alternating Diagonal Ordering Enclosing Standard Ordering ( in Brackets ) for a 5 x 6 Grid, (Al-Marhoun, 1980)

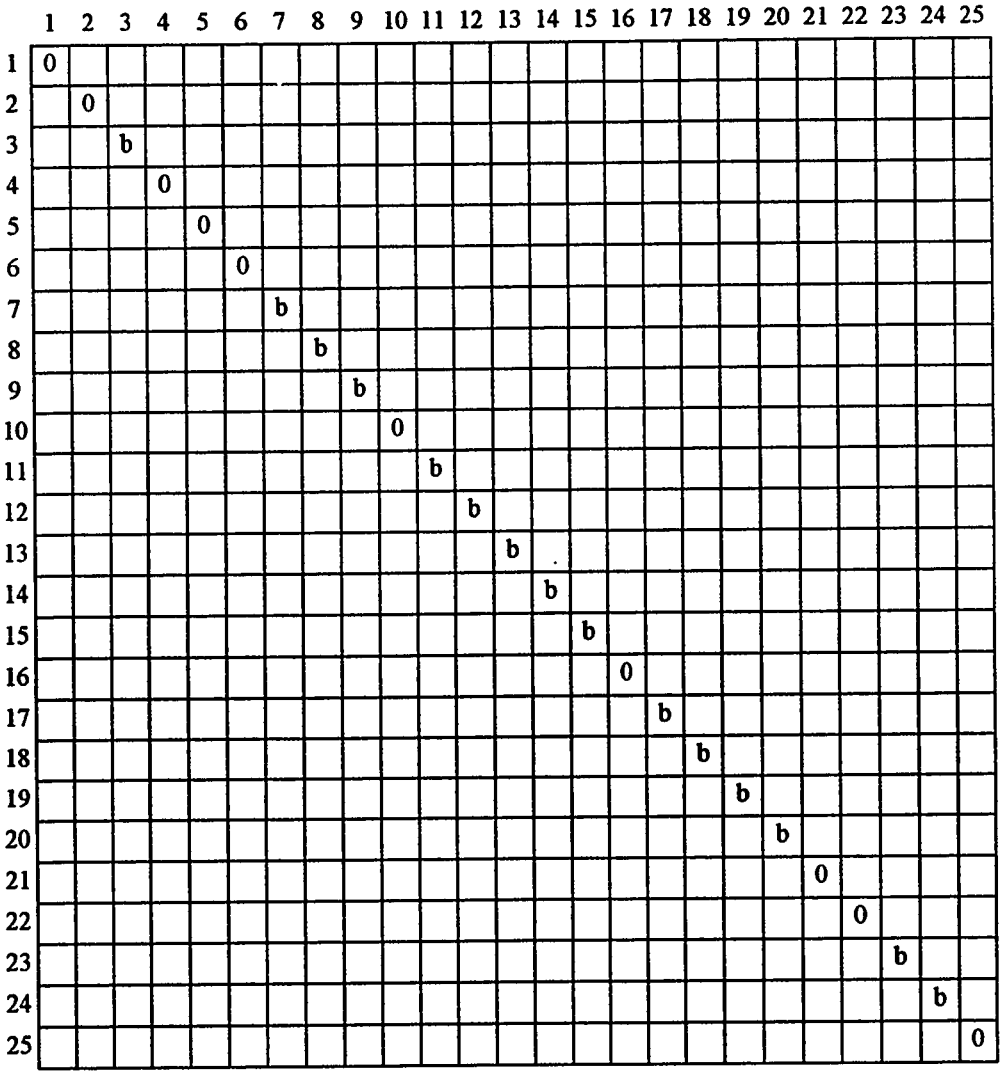



Figure 31- Upper Left Quadrant of the 50 x 50 Coefficient Matrix for the Grid Shown in Figure 30, Using Restricted Alternating Diagonal Ordering.

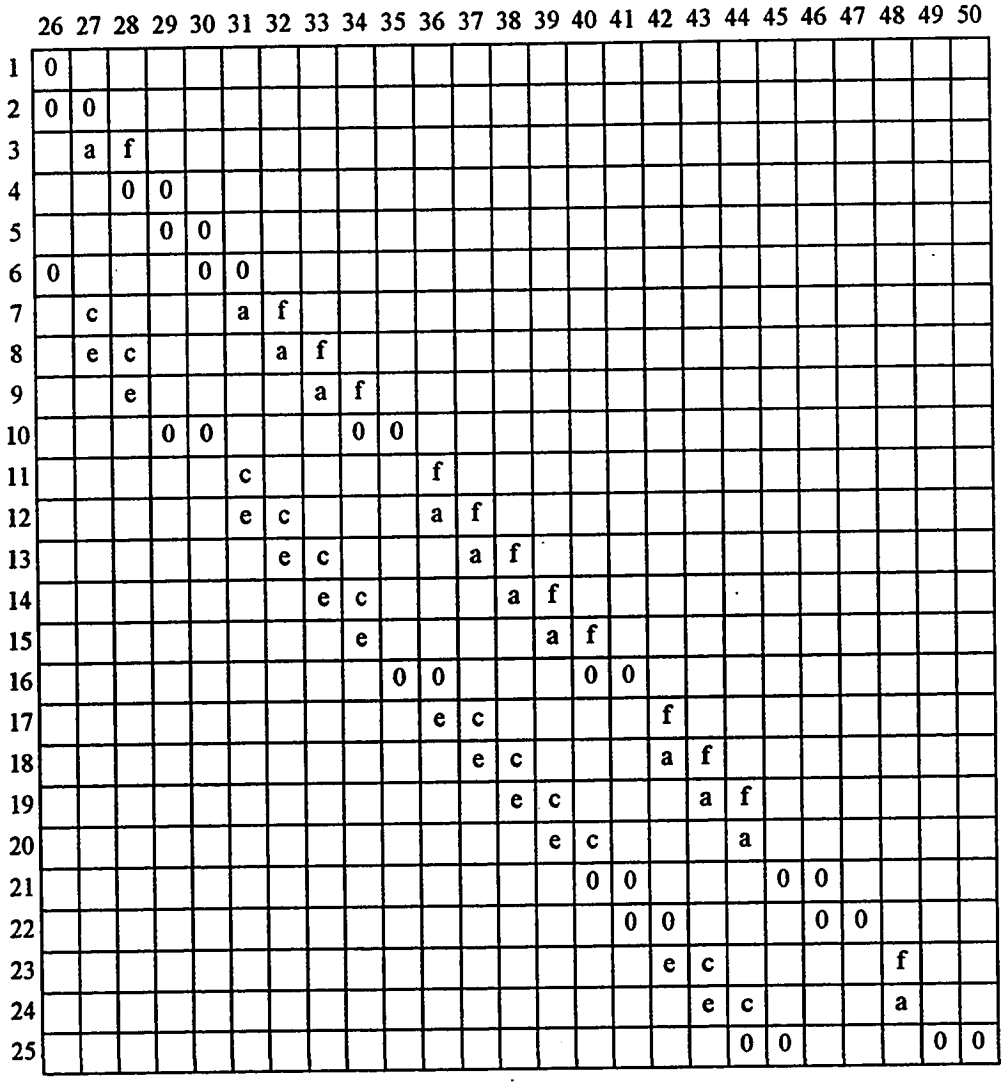
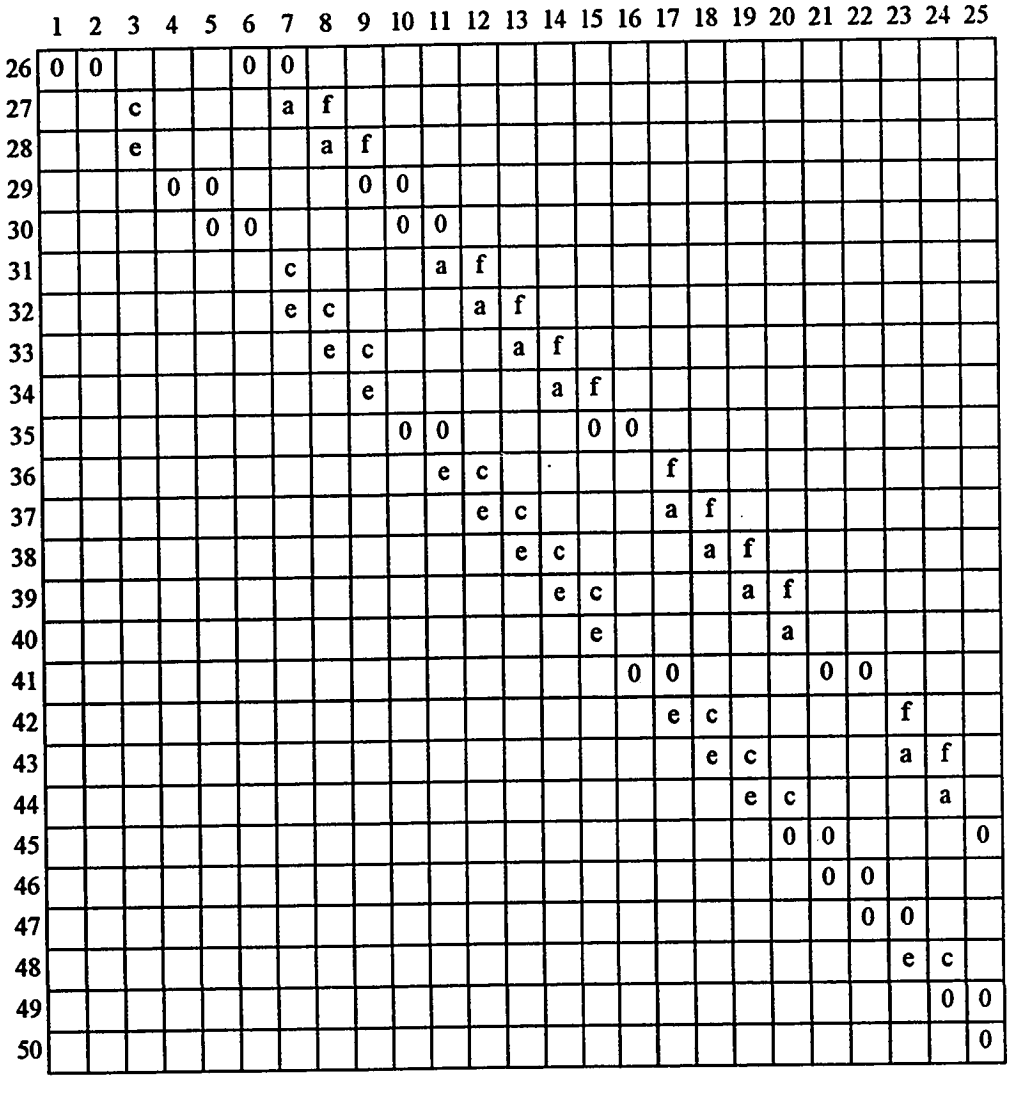



Figure 32 - Upper Right Quadrant of the 50 x 50 Coefficient Matrix for the Grid Shown in Figure 30, Using Restricted Alternating Diagonal Ordering.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 26 | 0 | 0 | | | | 0 | 0 | | | | | | | | | | | | | | | | | | |
| 27 | | | c | | | | a | f | | | | | | | | | | | | | | | | | |
| 28 | | | e | | | | | a | f | | | | | | | | | | | | | | | | |
| 29 | | | | 0 | 0 | | | | 0 | 0 | | | | | | | | | | | | | | | |
| 30 | | | | | 0 | 0 | | | | 0 | 0 | | | | | | | | | | | | | | |
| 31 | | | | | | | c | | | | a | f | | | | | | | | | | | | | |
| 32 | | | | | | | e | c | | | | a | f | | | | | | | | | | | | |
| 33 | | | | | | | | e | c | | | | a | f | | | | | | | | | | | |
| 34 | | | | | | | | | e | | | | | a | f | | | | | | | | | | |
| 35 | | | | | | | | | | 0 | 0 | | | | 0 | 0 | | | | | | | | | |
| 36 | | | | | | | | | | | e | c | | | | | f | | | | | | | | |
| 37 | | | | | | | | | | | | e | c | | | | a | f | | | | | | | |
| 38 | | | | | | | | | | | | | e | c | | | | a | f | | | | | | |
| 39 | | | | | | | | | | | | | | e | c | | | | a | f | | | | | |
| 40 | | | | | | | | | | | | | | | e | | | | | a | | | | | |
| 41 | | | | | | | | | | | | | | | | 0 | 0 | | | | 0 | 0 | | | |
| 42 | | | | | | | | | | | | | | | | | e | c | | | | | f | | |
| 43 | | | | | | | | | | | | | | | | | | e | c | | | | a | f | |
| 44 | | | | | | | | | | | | | | | | | | | e | c | | | | a | |
| 45 | | | | | | | | | | | | | | | | | | | | 0 | 0 | | | | 0 |
| 46 | | | | | | | | | | | | | | | | | | | | | 0 | 0 | | | |
| 47 | | | | | | | | | | | | | | | | | | | | | | 0 | 0 | | |
| 48 | | | | | | | | | | | | | | | | | | | | | | | e | c | |
| 49 | | | | | | | | | | | | | | | | | | | | | | | | 0 | 0 |
| 50 | | | | | | | | | | | | | | | | | | | | | | | | | 0 |

Figure 33- Lower Left Quadrant of the 50 x 50 Coefficient Matrix for the Grid Shown in Figure 30, Using Restricted Alternating Diagonal Ordering.

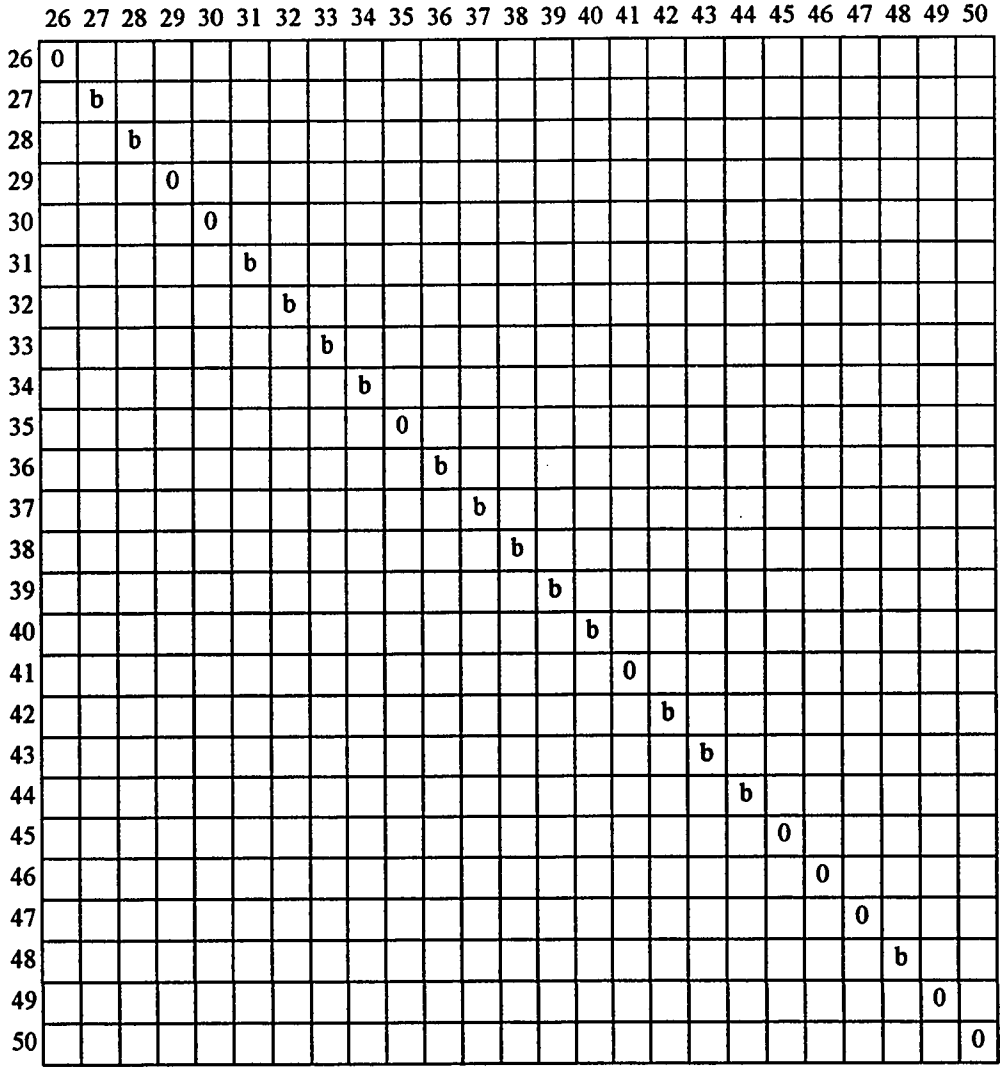



Figure 34 - Lower Right Quadrant of the 50 x 50 Coefficient Matrix for the Grid Shown in Figure 30, Using Restricted Alternating Diagonal Ordering.

| | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 26 | 1 | 1 | | | 1 | 1 | 1 | | | | | | | | | | | | | | | | | | |
| 27 | 0 | 1 | 1 | | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | | | | |
| 28 | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | | | |
| 29 | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | | |
| 30 | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | |
| 31 | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | | | | | |
| 32 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | | | | | | | | |
| 33 | | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | | | |
| 34 | | | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | | |
| 35 | | | | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | |
| 36 | | | | | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | | |
| 37 | | | | | | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | |
| 38 | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | |
| 39 | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | |
| 40 | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | |
| 41 | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | |
| 42 | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | |
| 43 | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | |
| 44 | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 45 | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 46 | | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 47 | | | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 48 | | | | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 49 | | | | | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 50 | | | | | | | | | | | | | | | | | | | 0 | 2 | 2 | 2 | 2 | 2 | 2 |

Figure 35 - The Lower Right Quadrant of the 50 x 50 Coefficient Matrix Showing New Elements Resulting from the Factorization Process ( The Number Indicate the Group of the Element ), ( Al-Marhoun, 1980 )

$$a'_j = \frac{a_j}{b_j} \tag{3.27}$$

$$c'_j = \frac{c_j}{b_j} \tag{3.28}$$

$$e'_j = \frac{e_j}{b_j} \tag{3.29}$$

$$f'_j = \frac{f_j}{b_j} \tag{3.30}$$

where $N_u = N/2$ if $N_d$ is even

$N_u = (N - m)/2$ if $N_d$ is odd

and $N_l = N - N_u$

$N_d$ = the number of diagonals in the system

m = semi-bandwidth = $L_d + 2$

$L_d$ = the length of the diagonal along which numbering is applied

For $j = 1, 2, \ldots, N_1$

$$G'_{1,j} = -e_{j+N_u} e'_j \tag{3.31}$$

$$G'_{2,j} = -e_{j+N_u} c'_j - c_{j+N_u} e'_{j+1} \tag{3.32}$$

$$G'_{3,j} = -c_{j+N_u} c'_{j+1} \tag{3.33}$$

$$G'_{i,j} = 0, \qquad i = 4, 5, \ldots, m-2 \tag{3.34}$$

$$G'_{m-1,j} = -e_{j+N_u} a'_j - a_{j+N_u} e'_{j+m-2} \tag{3.35}$$

$$G'_{m,j} = b_{j+N_u} - e_{j+N_u} f'_j - c_{j+N_u} a'_{j+1} - a_{j+N_u} c'_{j+m-2} - f_{j+N_u} e'_{j+m-1} \tag{3.36}$$

$$H'_{1,j} = -f_{j+N_u} f'_{j+m-1} \tag{3.37}$$

$$H'_{2,j} = -a_{j+N_u} f'_{j+m-2} - f_{j+N_u} a'_{j+m-1} \tag{3.38}$$

$$H'_{3,j} = -a_{j+N_u} a'_{j+m-2} \tag{3.39}$$

$$H'_{i,j} = 0, \qquad i = 4, 5, \ldots, m-2 \tag{3.40}$$

$$H'_{m-1,j} = -c_{j+N_u} f'_{j+1} - f_{j+N_u} c'_{j+m-1} \tag{3.41}$$

For the next computations refer to Figure 35.

**Group 0**

No more computation is needed.

**Group 1**

For $i = 1, 2, 3, \ldots, m$ calculate

$$H_{i,1} = H'_{i,1} / G_{m,1} \tag{3.42}$$

Then calculate for $j = 2, 3, \ldots, m-3$

$$G_{m,j} = G'_{m,j} - G_{m-1,j} H_{m-1,j-1} \tag{3.43}$$

$$H_{i,j} = \frac{1}{(G_{m,j})} \left( H'_{i,j} - H_{i-1,j-1}\, G_{m-1,j} \right) \qquad i = 2, 3, \ldots, j+1 \qquad (3.44)$$

$$H_{m-1,j} = H'_{m-1,j} / G_{m,j} \qquad (3.45)$$

**Group 2**

For $j = m-2, m-1, \ldots, N_l - 1$ calculate

$$G_{i,j} = G'_{i,j} - \sum_{k=1}^{i-1} \left( G_{i-k,j} H_{m-k,\, j-k-m+i} \right) \qquad i = 2, 3, \ldots, m \qquad (3.46)$$

$$H_{1,j} = H'_{1,j} / G_{m,j} \qquad (3.47)$$

$$H_{i,j} = \frac{1}{(G_{m,j})} \left( H'_{i,j} - \sum_{k=1}^{i-1} \left( H_{i-k,j-k}\ G_{m-k,j} \right) \right) \qquad (3.48)$$

for $i = 2, 3, \ldots, m-1$

Then calculate

$$G_{m,N_l} = G'_{m,N_l} - \sum_{k=1}^{m-1} \left( G_{m-k,N_l} H_{m-k,N_l-k} \right) \qquad (3.49)$$

Back Substitution

$$y_j = d_j / b_j, \qquad j = 1, 2, \ldots, N_u \qquad (3.50)$$

$$y_j = \frac{1}{(G_{m,j-N_u})}\left( d_j - e_j y_{j-N_u} - c_j y_{j-N_u+1} - a_j y_{j-N_u+m-2} - f_j y_{j-N_u+m-1} \right)$$

$$- \sum_{k=1}^{m-1} \left( G_{k,j-N_u} y_{j-m+k} \right) \quad j = N_u + 1, \ldots, N \tag{3.51}$$

$$P_N = y_N$$

$$P_j = y_j - \sum_{k=1}^{m-1} \left( H_{k,j-N_u} P_{j+m-k} \right) \qquad j = N-1, N-2, \ldots, N_u + 1 \tag{3.52}$$

$$P_j = y_j - f'_j P_{j+N_u} - a' P_{j-1+N_u} - c'_j P_{j-m+2+N_u} - e'_j P_{j-m+1+N_u} \tag{3.53}$$

$$j = N_u, N_u - 1, \ldots, 1$$

# CHAPTER 4

# ITERATIVE METHODS

It is often more economical in terms of computer time and storage to use an iterative technique rather than the direct elimination techniques. Iterative methods usually require a set of parameters to accelerate their convergence to an acceptable level. The methods can be divided into two categories : (1) point iterative, and (2) block iterative [13]. In the point iterative methods the computation is performed for every single point successively, and thus no matrix computation is involved. Their application is therefore restricted to relatively simple cases in reservoir simulation problems. The block iterative methods on the other hand perform the computation for a particular block of cells that are easy to solve and therefore require some form of matrix computation. Both classes of methods must be written in the matrix form to be solved.

Most of the iterative methods are of the form

$$\mathbf{P}^{(\nu+1)} = \mathbf{B}\,\mathbf{P}^{(\nu)} + \mathbf{k} \qquad (4.1)$$

By making use of the solution from several previous iteration levels, the above procedure can be generalized in the following form :

$$\upsilon^{(\nu)} = \mathbf{B}\,\upsilon^{(\nu-1)} + \mathbf{k} \qquad \nu = 1, 2, \ldots, \quad \text{and } \upsilon^{(0)} \text{ is arbitrary}$$

$$\mathbf{P}^{(\nu)} = b_{\nu}\,\upsilon^{(\nu)} + \sum_{l=0}^{\nu-1} (b_{l,\nu}\,\mathbf{P}^{(l)}) \tag{4.2}$$

The iterative methods differ in the approaches employed for the selection of the b values in equation 4.2 above. These approaches include minimized iterations, conjugate gradients and steepest descents.

Another class of methods has been proposed by Meijerink and Van der Vorst [18]. Their approach is similar to that used by Stone [2] . They are based on incomplete **LU** decomposition and combine some of these methods with conjugate gradient methods.

There are varieties of iterative methods available. Some of those methods that are of common use in reservoir simulation will be discussed in the following sections. These are :

(1) Point Successive Over-Relaxation, **PSOR**

(2) Line Successive Over-Relaxation , **LSOR**

(3) Alternating Direction Implicit Procedure, **ADIP**

(4) Strongly Implicit Procedure, **SIP**

(5) Conjugate Gradient-Truncated Direct Method, **CGTD**

(6) Nested Factorization, **NF**

## 4.1 Convergence

Convergence of iterative methods will be discussed before discussion of the actual methods. Consider a matrix equation of the form

$$\mathbf{A}\,\mathbf{u} = \mathbf{d}$$

where **A** is the coefficient matrix. Any iterative scheme for this problem may be expressed as

$$\mathbf{u}^{(\nu+1)} = \mathbf{B}\,\mathbf{u}^{(\nu)} + \mathbf{b}$$

where the matrix **B** and the vector **b** depend upon the problem being solved and the method of iteration. If **B** is independent of $\nu$ then the iterative scheme is called stationary.

An iteration scheme is convergent if

$$\lim_{\nu \to \infty} u^{(\nu)} = u$$

where **u** is the exact solution. Obviously it is necessary ( but not sufficient !) [19] for convergence of a scheme that it satisfies the condition

$$\mathbf{u} = \mathbf{B}\,\mathbf{u} + \mathbf{b}$$

If the error at various stages of iteration is defined as

$$\varepsilon^{(\nu)} = \mathbf{u}^{(\nu)} - \mathbf{u}$$

then by subtracting

$$\mathbf{u}^{(\nu)} = \mathbf{B}\mathbf{u} + \mathbf{b}$$

from

$$\mathbf{u}^{(\nu+1)} = \mathbf{B}\mathbf{u}^{(\nu)} + \mathbf{b}$$

we get

$$\mathbf{u}^{(\nu+1)} - \mathbf{u}^{(\nu)} = \mathbf{B}(\mathbf{u}^{(\nu)} - \mathbf{u})$$

By defining

$$\varepsilon^{(\nu+1)} = \mathbf{u}^{(\nu+1)} - \mathbf{u}^{(\nu)}$$

then

$$\varepsilon^{(\nu+1)} = \mathbf{B}\varepsilon^{(\nu)}$$

or, in terms of the initial error vector $\varepsilon^{(0)}$,

$$\varepsilon^{(\nu)} = \mathbf{B}^{\nu}\varepsilon^{(0)}$$

In order that the equation

$$\mathbf{u}^{(\nu+1)} = \mathbf{B}\mathbf{u}^{(\nu)} + \mathbf{b}$$

be convergent $\varepsilon^{(\nu)}$ must approach zero for arbitrary $\varepsilon^{(0)}$. This implies that $\| \mathbf{B}^{\nu} \|$ must approach zero. Therefore, the iteration scheme

$$\mathbf{u}(\nu+1) = \mathbf{B}\,\mathbf{u}(\nu) + \mathbf{b}$$

converges if and only if

$$\rho(\mathbf{B}) < 1$$

where $\rho(\mathbf{B})$ is the spectral radius of **B** and is defined as

$$\rho(\mathbf{B}) = \max_i |\lambda_i|$$

and $\lambda_i$ is the eigenvalue.

The rate of convergence is also related to $\rho(\mathbf{B})$. The ratio of the norm of the error at ( $\nu$ ) level to that at (0 ) level is as follows

$$\frac{\|\varepsilon^{(\nu)}\|}{\|\varepsilon^{(0)}\|} \leq \|\mathbf{B}^{\nu}\|$$

and $\|\mathbf{B}^{\nu}\|$ serves as a comparison of different iterative methods [ 19 ].

## 4.2 Stopping Criteria

Certain stopping criteria are needed for the iterative techniques. One criterion is to iterate untill a certain tolerance is met as follows

$$\|P_i^{\nu} - P_i^{\nu-1}\| < \varepsilon$$

where $\varepsilon > 0$. This tolerance will be dependent on the problem being solved and the degree of accuracy desired. Another stopping criterion is to use the relative error as follows :

$$\frac{\| P_i^{\nu} - P_i^{\nu-1} \|}{\| P_i^{\nu} \|} < \varepsilon$$

Again, the tolerance that should be used is dependent on the degree of accuracy required and the problem being solved.

Another stopping criterion is to calculate the residual vector. If $\tilde{\mathbf{P}}$ is an approximation to the solution $\mathbf{P}$ of $\mathbf{A P = d}$ the residual vector can be calculated as $\mathbf{r = A \tilde{P} - d}$. This residual can be checked aagainst a certain tolerance which is also dependent on the problem being solved and the accuracy desired. However, in certain cases even if $\| \mathbf{r} \|$ is small, this does not mean that $\| \mathbf{P} - \tilde{\mathbf{P}} \|$ would be small as well since it will also be dependent on the matrix $\mathbf{A}$ as well[21].

## 4.3 Successive Over-Relaxation Methods

Consider a system of equations in matrix form

$$\mathbf{A P = d} \tag{4.3}$$

Writing the above equation for each individual cell in an individual row will results into a number of equations as follows:

$$P_1 = \frac{1}{a_{11}}[\, b_1 - (a_{12} P_2 + a_{13} P_3 + \ldots + a_{1n} P_n)\,]$$

$$P_2 = \frac{1}{a_{22}}[\, b_2 - (a_{21} P_1 + a_{23} P_3 + \ldots + a_{2n} P_n)\,]$$

$$\vdots$$

$$P_n = \frac{1}{a_{nn}}[\, b_n - (a_{n1} P_1 + a_{n2} P_2 + \ldots + a_{n,\,n-1} P_n)\,] \qquad (4.4)$$

Knowing an original estimate of all the unknown quantities, we can sequentially use the equations (4.4) and obtain new approximations to the solution vector **P**. This approach can be speeded up to achieve varying rate of convergence. The successive relaxation methods use a technique where not only the most recent values of $\mathbf{P_i}$ are used in the solution process but rather modified values are being substituted into the equation. The modification involves the addition of a quantity to the unknown values of $\mathbf{P_i}$ before substituting them into equations 4.4. This relaxation process is as follows:

$$P_i^{\nu} = P_i^{\nu-1} + \omega \frac{r_i^{\nu-1}}{a_{ii}} \qquad \text{for } i = 1, 2, \ldots, n \qquad (4.5)$$

where the residual $r_i$ is calculated as

$$r_i^{\nu-1} = b_i - \sum a_{ij} P_i^{\nu-1} \qquad \text{for } i = 1, 2, \ldots, n \qquad (4.6)$$

The factor $\omega$ in equation (4.5), the relaxation parameter, accelerates the convergence process. It is found that for any fixed value of $\omega$ between 0 and 2 the process converges for the type of equations encountered in reservoir simulation [16]. The optimum $\omega_{opt}$ produces extremely rapid convergence.

The selection of this optimum value requires finding the relationship between the number of iterations required for convergence against a constant tolerance for varying relaxation parameters. Several simulation runs are made and the data is plotted graphically as is shown in Figure 36. The curve has a characteristic shape, and the lowest point on the curve corresponds to the best value of the relaxation parameter. Generally, the value of $\omega$ approaches 1.0 the more homogeneous the system is. The value approaches 2.0 the more anisotropic the system is. When $\omega$ approaches $\omega_{opt}$ from below, the curve has an almost vertical tangent, and a value of $\omega$ chosen just too small causes poor convergence. The slope beyond the $\omega_{opt}$ is close to linear. Hence, a value just above does not affect the convergence as radically. In practice, an overestimation is always better than an underestimation [16].

A sufficient condition for the convergence of **SOR** methods is that the real matrix **A** with positive diagonal entries, be irreducibly diagonally dominant. This condition is easily met in most reservoir engineering problems. The only difficulty encountered is in the determination of the over-relaxation parameter $\omega$ [ 4 ].

### 4.3.1 Point Successive Over-Relaxation, PSOR

This method involves the application of the relaxation technique to individual cells in the simulation grid sequentially.

The general form of the system of linearized equations is as follows :

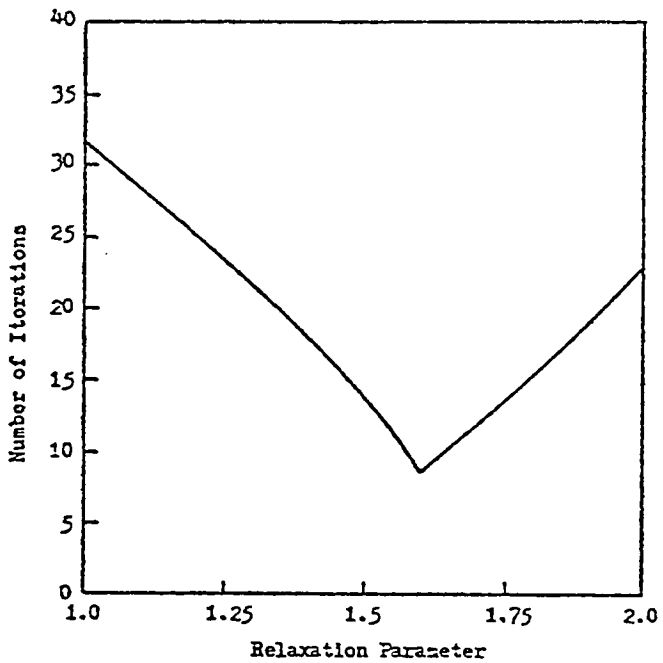



Figure 36 - A Typical Plot of Relaxation Parameter at Fixed Tolerence

$$e_i P_{i-\eta} + a_i P_{i-1} + b_i P_i + c_i P_{i+1} + f_i P_{i+\eta} = d_i \quad (4.7)$$

The above formula is solved for each cell as follows :

$$\tilde{P}_i^{\nu+1} = \frac{1}{b_i} \left( d_i - e_i P_{i-\eta}^{\nu+1} - a_i P_{i-1}^{\nu+1} - c_i P_{i+1}^{\nu} - f_i P_{i+\eta}^{\nu} \right) \quad (4.8)$$

After calculating for each point, the calculated value of $\tilde{P}_i^{\nu+1}$ is relaxed as follows :

$$P_i^{\nu+1} = P_i^{\nu} + \omega \left( \tilde{P}_i^{\nu+1} - P_i^{\nu} \right) \quad (4.9)$$

The maximum absolute relative error of successive iteration is used as the convergence criterion. It must be within a pre-defined maximum absolute tolerance ( $\varepsilon$ ) as follows :

$$r_{max} \leq \varepsilon \quad (4.10)$$

$$r_{max} = \max |r_i| \quad (4.11)$$

and

$$r_i = \frac{(P_i^{\nu} - P_i^{\nu-1})}{P_i^{\nu}} \quad (4.12)$$

The $\omega$ is the relaxation parameter. A flowchart of this method is shown in Figure 37 .

### 4.3.2 Line Successive Over-Relaxation, LSOR

The real power of the **SOR** method for reservoir simulation lies in its application as a line **SOR** or block **SOR** method. This method treats the two dimensional reservoir problem as a series of one dimensional problems. Instead of computing the solution to an individual cell at a time, a block of cells, usually a row, is solved at a time. This method relaxes one line at a time and that is why it is called as such. Assuming that the values on the previous line, - i.e. $i - \eta$ terms - are known, and the values at $( i + \eta )$ are approximated by the old iteration values, then equation (4.7) above can be solved for those grids in a particular line as follows :

$$a_i \tilde{P}^{\nu+1}_{i-1} + b_i \tilde{P}^{\nu+1}_{i} + c_i \tilde{P}^{\nu+1}_{i+1} =$$

$$d_i - e_i \tilde{P}^{\nu+1}_{i-\eta} - f_i \tilde{P}^{\nu}_{i+\eta} \qquad (4.13)$$

Equation (4.13) can be simplified to

$$a_i \tilde{P}^{\nu+1}_{i-1} + b_i \tilde{P}^{\nu+1}_{i} + c_i \tilde{P}^{\nu+1}_{i+1} = d'_i \qquad (4.14)$$

where

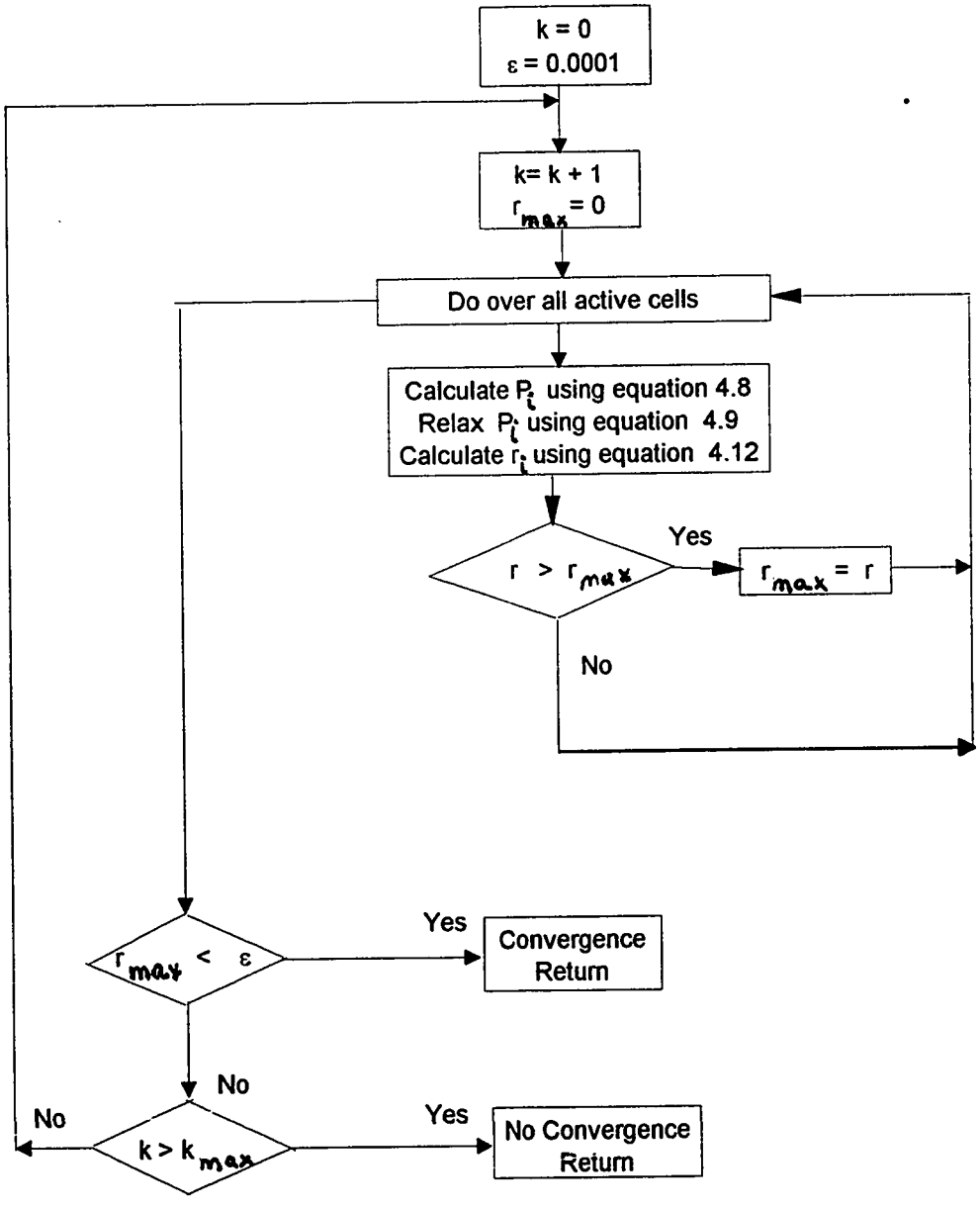



Figure 37 - PSOR Flow Chart

$$d'_i = d_i - e_i \tilde{P}^{\nu+1}_{i-\eta} - f_i \tilde{P}^{\nu}_{i+\eta} \qquad (4.15)$$

Equation (4.14) forms a close-band tridiagonal matrix which can be solved readily utilizing the close-band Thomas algorithm or the Gaussian elimination process. The value of $P_i$ obtained for every cell of that row is relaxed as follows :

$$P^{k+1}_i = P^k_i + \omega (\tilde{P}^{k+1}_i - P^k_i) \qquad (4.16)$$

The flow chart for **LSOR** is shown in Figure 38.

## 4.4 Thomas Algorithm

This algorithm can be used to solve the close-band tridiagonal matrix generated in **LSOR** or **ADIP** methods. The equations are

$$a_i P_{i-1} + b_i P_i + c_i P_{i+1} = d_i$$

$$\text{for } 1 \leq i \leq N \qquad \text{with } a_1 = a_N = 0 \qquad (4.17)$$

The algorithm is as follows :

For $1 \leq i \leq N$, compute

$$\zeta_i = b_i - \frac{a_i c_{i-1}}{\zeta_{i-1}} \qquad \text{with } \zeta_1 = b_1 \qquad (4.18)$$

and

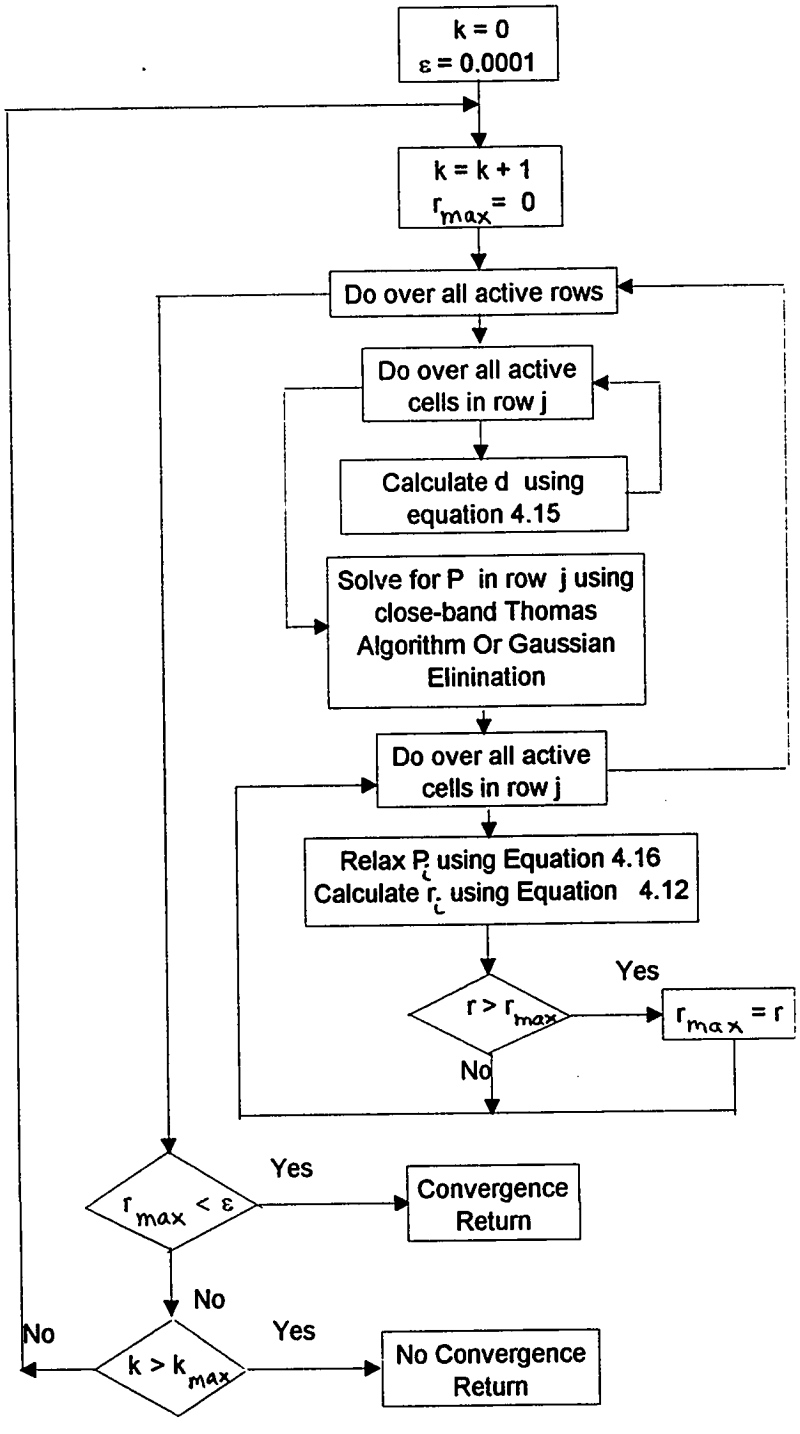



Figure 38 - LSOR Flow Chart

$$\xi_i = \frac{d_i - a_i \xi_{i-1}}{\zeta_i} \qquad \text{with } \xi_1 = \frac{d_1}{b_1} \qquad (4.19)$$

The unknown pressures are then computed as

$$P_N = \xi_N$$

and

$$P_i = \xi_i - \frac{c_i P_{i+1}}{\zeta_i} \qquad \text{for } i = N-1, N-2, \ldots, 1 \qquad (4.20)$$

The flow chart of the Thomas algorithm is shown in Figure 39.

## 4.5 Alternating Direction Implicit Procedure

This method treats the reservoir problem as a succession of one dimensional problems in both row and column directions. The iteration step is divided into two equal substeps. During the first step the cells are swept horizontally, i.e. every row, solving for the unknown pressures. In the second step the cells are swept vertically, i.e. every column. Crichlow [16] describes the method as follows. Given the general equaion for a certain cell ,

$$e_i P_{i-\eta}^{n+1} + a_i P_{i-1}^{n+1} + b_i P_i^{n+1} + c_i P_{i+1}^{n+1} + f_i P_{i+\eta}^{n+1} = d_i \qquad (4.21)$$

In the first half of the time step, the x-direction sweep is carried out. $P_{i+\eta}^{n+1}$ is assumed known from the previous time step value and $P_{i-\eta}^{n+1}$ has just

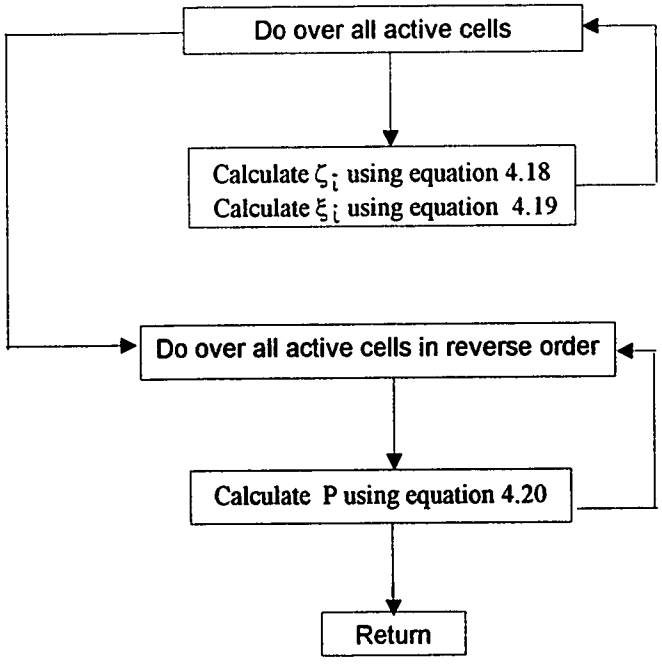



Figure 39 - Flow Chart for Close - Band Thomas Algorithm

been calculated. Now there are only three unknowns instead of five. Equation (4.21) becomes

$$a_i P_{i-1}^{n+(1/2)} + b_i P_i^{n+(1/2)} + c_i P_{i+1}^{n+(1/2)}$$

$$= d_i - e_i P_{i-\eta}^{n+(1/2)} - f_i P_{i+\eta}^{n+(1/2)} \qquad (4.22)$$

This equation when written for each cell in the model produces a close-band tridiagonal matrix as is shown in Figure 40. The individual small 5 x 5 matrices are solved using Gaussian elimination or the Thomas algorithm as was explained previously.

In the second half of the time step the solution is carried out in the vertical direction; i.e. sweeping every column. $P_{i+1}^{n+1}$ is assumed known from the previous time step value and $P_{i-1}^{n+1}$ has just been calculated. Now with only three unknowns instead of five, Equation (4.22) becomes

$$e_i P_{i+\eta}^{n+1} + b_i P_i^{n+1} + f_i P_{i+\eta}^{n+1}$$

$$= d_i - a_i P_{i-1}^{n+1} - c_i P_{i+1}^{n+(1/2)} \qquad (4.23)$$

This equation when written for each cell in the model produces a wide-band tridiagonal matrix as is shown in Figure 41. The individual small 5 x 5 mtrices are solved using Gaussian elimination or Thomas algorithm as was done in the horizontal sweep.

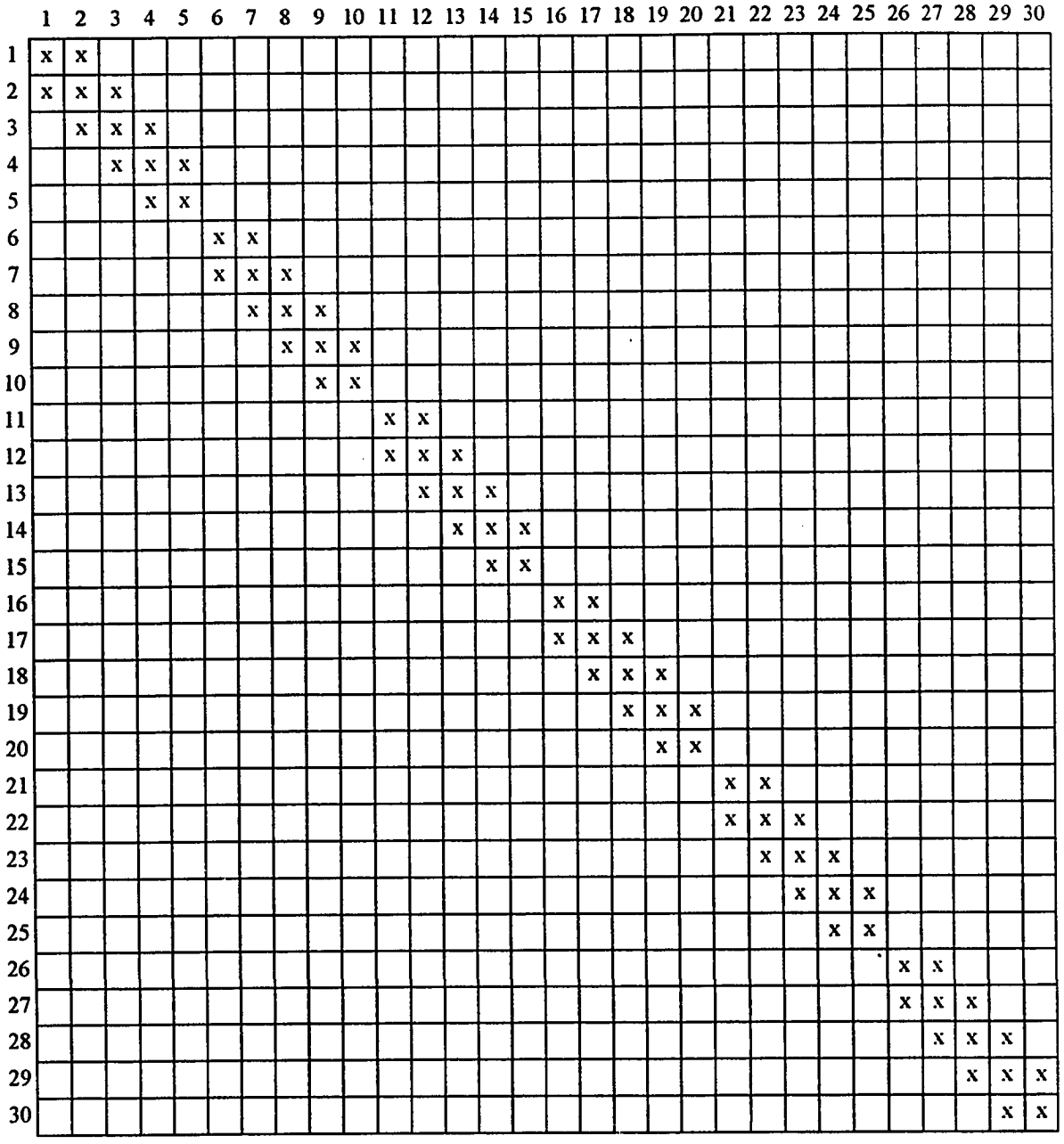



Figure 40 - 30 x 30 Coefficient Matrix for Individual Rows for the 5 x 6 Grid in Figure - 19, Using Standard Ordering.

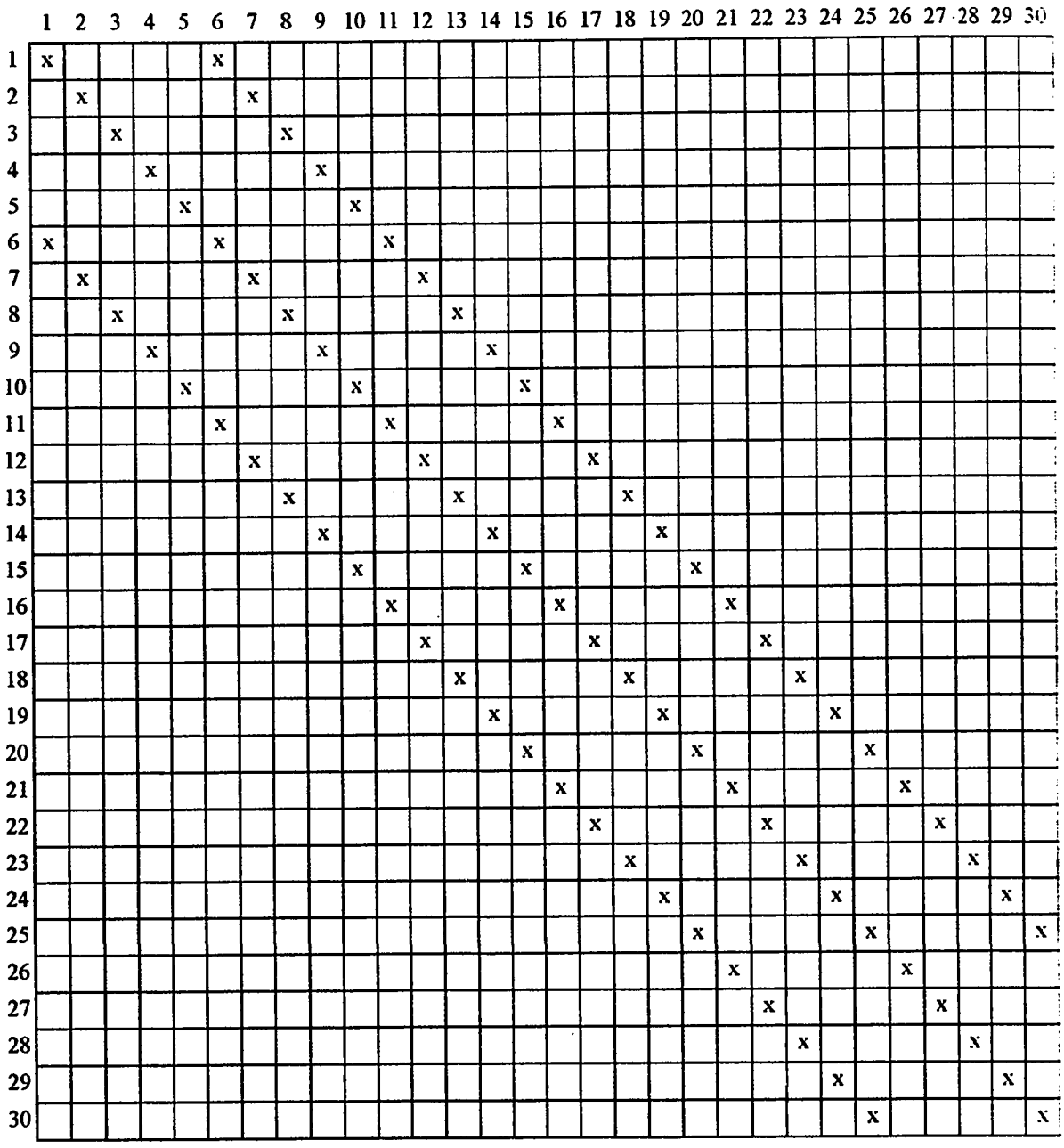


Figure 41 - 30 x 30 Coefficient Matrix for the 5 x 6 Grid in Figure - 19, for Individual Columns when Ordering by Rows is Used.

**ADIP** can be improved by the use of a modifier called an acceleration parameter. The above **ADI** scheme is repeated several times in a given time step untill there is convergence. This scheme is described by Aziz et al. as follows [ 19 ].

Given the general equation for a certain cell,

$$e_i P_{i-\eta} + a_i P_{i-1} + b_i P_i + c_i P_{i+1} + f_i P_{i+\eta} = d_i$$

By separating the contribution to $b_i$ from the two space derivatives, we get

$$[\, a_i P_{i-1} + b x_i P_i + c_i P_{i+1} \,]$$

$$+ [\, e_i P_{i-\eta} + b y_i P_i + f_i P_{i+\eta} \,]$$

$$+ [\varphi u_i] = d_i \qquad (4.24)$$

where all the coefficients have the same meaning as in equation (4.7) above.

The above equation can be separated into three parts, one for the contribution to the cell in the horizontal direction, $\delta/\delta x[\lambda x(\delta U/\delta x)]$, the second the contribution to the cell in the vertical direction $\delta/\delta y[\lambda y(\delta U/\delta y)]$, and the third is the contribution from the time derivative as follows:

$$\delta/\delta x[\lambda x(\delta U/\delta x)] = [\, a_i P_{i-1} + b x_i P_i + c_i P_{i+1} \,] \qquad (4.25)$$

$$\delta/\delta y[\lambda y(\delta U/\delta y)] = [\, e_i P_{i-\eta} + b y_i P_i + f_i P_{i+\eta} \,] \qquad (4.26)$$

$$\text{contribution from time derivative} = [\varphi u_i] \qquad (4.27)$$

Equation (4.17) can be put in a matrix form as follows

$$(\mathbf{H} + \mathbf{V} + \sum)\mathbf{u} = \mathbf{d} \qquad (4.28)$$

If the standard ordering shown in Figure 19 is used, then **H** and **V** will take the forms shown in Figures 40 and 41 respectively, when ordering by row is performed. If ordering is performed by column then the matrix **V** will take the shape shown in Figure 42. The $\sum$ is a diagonal matrix containing $\varphi_i$. It is clear from these figures that an equation of the form

$$\mathbf{H}\,\mathbf{u} = \mathbf{k} \qquad (4.29)$$

$$\mathbf{V}\,\mathbf{u} = \mathbf{k} \qquad (4.30)$$

could be solved by a repeated application of some tridiagonal matrix solution algorithm, such as Thomas algorithm, or simply by applying the Gaussian elimination procedure.

Then equation (4.28) can be written as

$$(\mathbf{H} + \tfrac{1}{2}\sum + r\mathbf{I})\mathbf{u} = (r\mathbf{I} - \mathbf{V} - \tfrac{1}{2}\sum)\mathbf{u} + \mathbf{d} \qquad (4.31)$$

or

$$(\mathbf{V} + \tfrac{1}{2}\sum + r\mathbf{I})\mathbf{u} = (r\mathbf{I} - \mathbf{H} - \tfrac{1}{2}\sum)\mathbf{u} + \mathbf{d} \qquad (4.32)$$

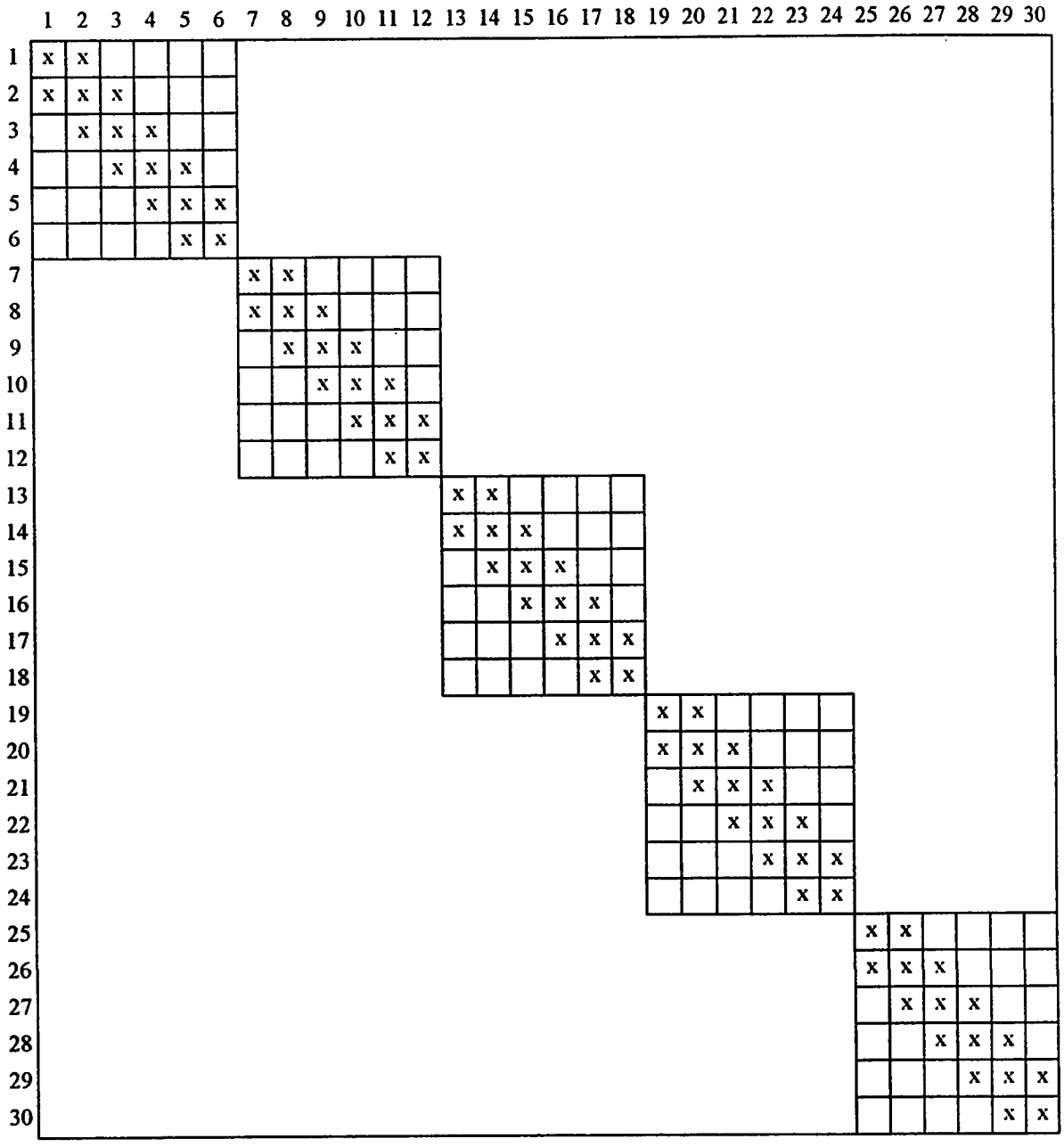



Figure 42 - 30 x 30 Coefficient Matrix for Individual Columns for the 5 x 6 Grid in Figure - 19, when Ordering is by Column.

where r is some positive scalar used as an iteration parameter chosen to accelerate convergence of the iterative process. Both of the above equations can be solved readily. If we define

$$\mathbf{H}_1 = \mathbf{H} + \frac{1}{2}\Sigma \tag{4.33}$$

and

$$\mathbf{V}_1 = \mathbf{V} + \frac{1}{2}\Sigma \tag{4.34}$$

then the following iterative process can be used

$$(\mathbf{H}_1 + r^{(\nu+1)}\mathbf{I})\,\mathbf{u}^* = (r^{(\nu+1)}\mathbf{I} - \mathbf{V}_1)\,\mathbf{u}^{(\nu)} + \mathbf{d} \tag{4.35}$$

$$(\mathbf{V}_1 + r^{(\nu+1)}\mathbf{I})\,\mathbf{u}^{(\nu+1)} = (r^{(\nu+1)}\mathbf{I} - \mathbf{H}_1)\,\mathbf{u}^* + \mathbf{d} \tag{4.36}$$

$$\nu = 0, 1, 2, \ldots$$

where $\mathbf{u}^{(0)}$ is an arbitrary approximation of the solution vector **u**. Equation (4.35) is solved first by considering unknowns for each line in the x-direction. The final result of this step is $\mathbf{u}^*$. After completing this sweep, the sweep is altered into the vertical direction and solved for $\mathbf{u}^{(\nu+1)}$ from equation (4.36) by considering unknowns along each line in the y-direction. This process is continued untill convergence is obtained. A flow chart of this procedure is shown in Figure 43.

The power of these methods can only be realized by using a sequence of parameters $r^{(m)}$ in a cycle and repeating the cycle untill $\mathbf{u}^{(\nu)}$ converges to **u**. This points out the major problem in the application of **ADI** methods : What is the

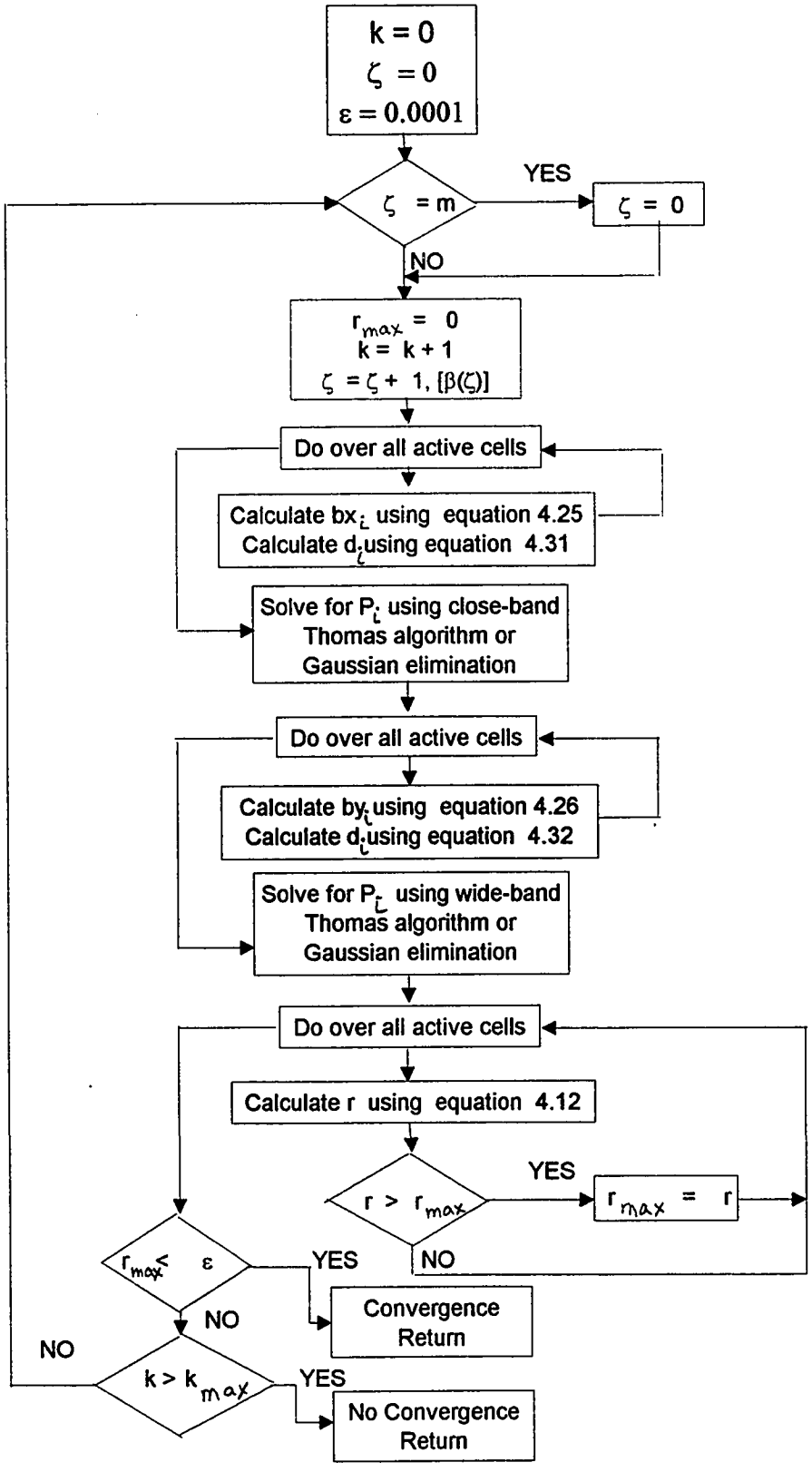



Figure 43 - ADIP Flow Chart.

optimum number of iterations in a cycle and what are the optimum parameters for the cycle? There are many practical methods for the selection of iteration parameters. Two of those procedures which are in common use are discussed next.

## 1. Estimation of Iteration Parameters by the Peaceman - Rachford Method.

Let a be the lower bound for the eigenvalues of **V** and b the upper bound for the eigenvalues of **H**. The procedure is as follows (13)

1. Estimate a , b , and compute c = a / b.
2. Find the smallest integer M such that $( 0.414 )^{2M} \leq c$ .
3. Calculate $r^{(m)}$ as

$$r^{(m)} = b\,(c)^{(2m-1)/2M} \qquad m = 1, 2, \ldots, M \qquad (4.37)$$

4. Estimate the asymptotic rate of convergence from

$$R(\mathbf{ADI}) = -\frac{2}{M} \ln \left[ \frac{1 - c^{1/(2M)}}{1 + c^{1/(2M)}} \right] \qquad (4.38)$$

## 2. Estimation of Iteration Parameters by the Wachspress Method.

1. Estimate a, b, and compute c = a / b.
2. Find the smallest integer M such that $( 0.172 )^{M-1} \leq c$ .
3. Calculate $r^{(m)}$ from a geometric sequence :

$$r^{(m)} = b\,(c)^{(m-1)/(M-1)} \qquad M > 2; \quad m = 1, 2, \ldots, M \qquad (4.39)$$

4. Estimate the asymptotic rate of convergence from

$$R(\mathbf{ADI}) = -\frac{2}{M} \ln \left[ \frac{1 - c^{(1/2)(M-1)}}{1 + c^{(1/2)(M-1)}} \right] \qquad (4.40)$$

The Wachspress method is considered superior for most practical problems. In many practical cases it is difficult to estimate a and b. For such cases the largest value of $r^{(m)}$ is selected close to 1 and the parameters distributed in a geometric sequence as follows:

$$r^{(1)} = r_{min} \qquad r^{(M)} = r_{max}$$

$$\frac{r^{(m+1)}}{r^{(m)}} = \gamma \qquad m = 1, 2, \ldots, M-1$$

$$\gamma = \left[ \frac{r_{max}}{r_{min}} \right]^{1/M-1}$$

The above method is the same as equation (4.39) based on the work of Wachspress with $r_{min} = a$ and $r_{max} = b$. The lower end of the sequence and M are selected by trial and error [13].

## 4.6 Strongly Implicit Procedure (SIP)

This method was developed by Stone [2]. This procedure uses an iterative

technique which has a significantly higher rate of convergence and which is not sensitive to the selection of iteration parameters. The basic idea of the strongly implicit procedure ( **SIP** ) is to alter the original matrix **A** and solve the system of simultaneous linear equations by an elimination process working on the modified version of the original matrix. This modified matrix is more easily factorable into an **LU** product which has only three nonzero elements in each row; this minimizes the work required to solve the system by elimination.

As discussed earlier, the matrix **A** can be solved by direct elimination by factoring it into the product of a lower triangular matrix **L** and an upper triangular matrix **U**. In general **L** will have non-zero elements from the main diagonal to the diagonal corresponding to the diagonal of e values, Figure 44. Similarly, the matrix **U** will have non-zero elements from the main diagonal to the diagonal corresponding to f values, Figure 45. As indicated by Stone, for each grid point approximately I + J such elements must be computed which makes the elimination process slow for large I and J.

The development of this procedure is as follows. Consider equation (4.3)

$$\mathbf{A\ P = d}$$

and add **NP** to both sides, and add and subtract **AP** from the right side to yield

$$\mathbf{(A+N)P = (A+N)P - (AP - d)} \tag{4.41}$$

The method proposed by Stone is to find **N** so that **A + N** is easily factored into a product **LU**, such that **L** and **U** have only three non-zero elements in each row as shown in Figures 44 and 45 for the grid of Figure 19, [13]. The elements of **L** and **U** matrices may be computed recursively from

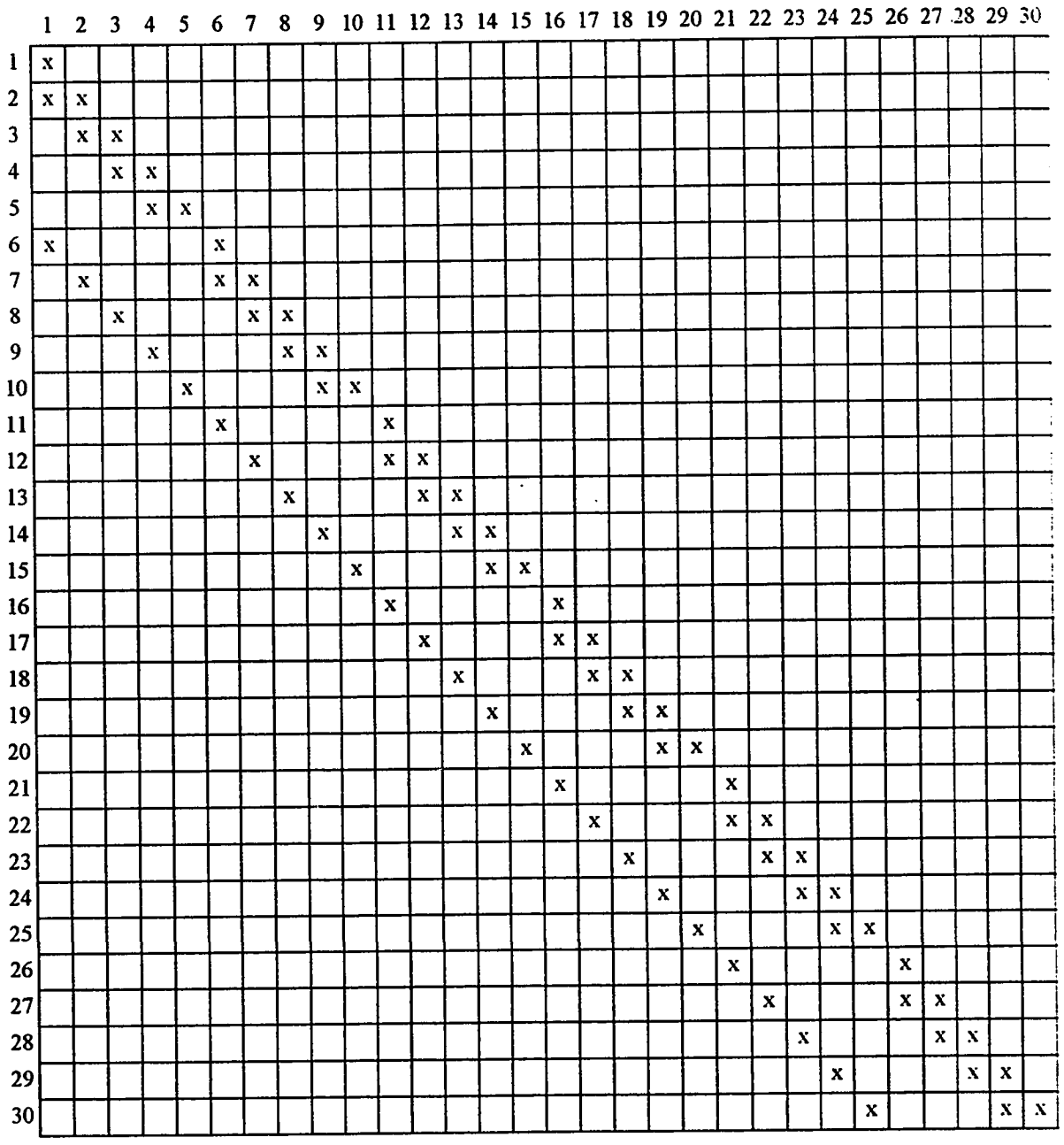



Figure 44 - Form of the Lower Matrix ( **L** ) for **SIP** method.
Non-zero Entries are Indicated by x.

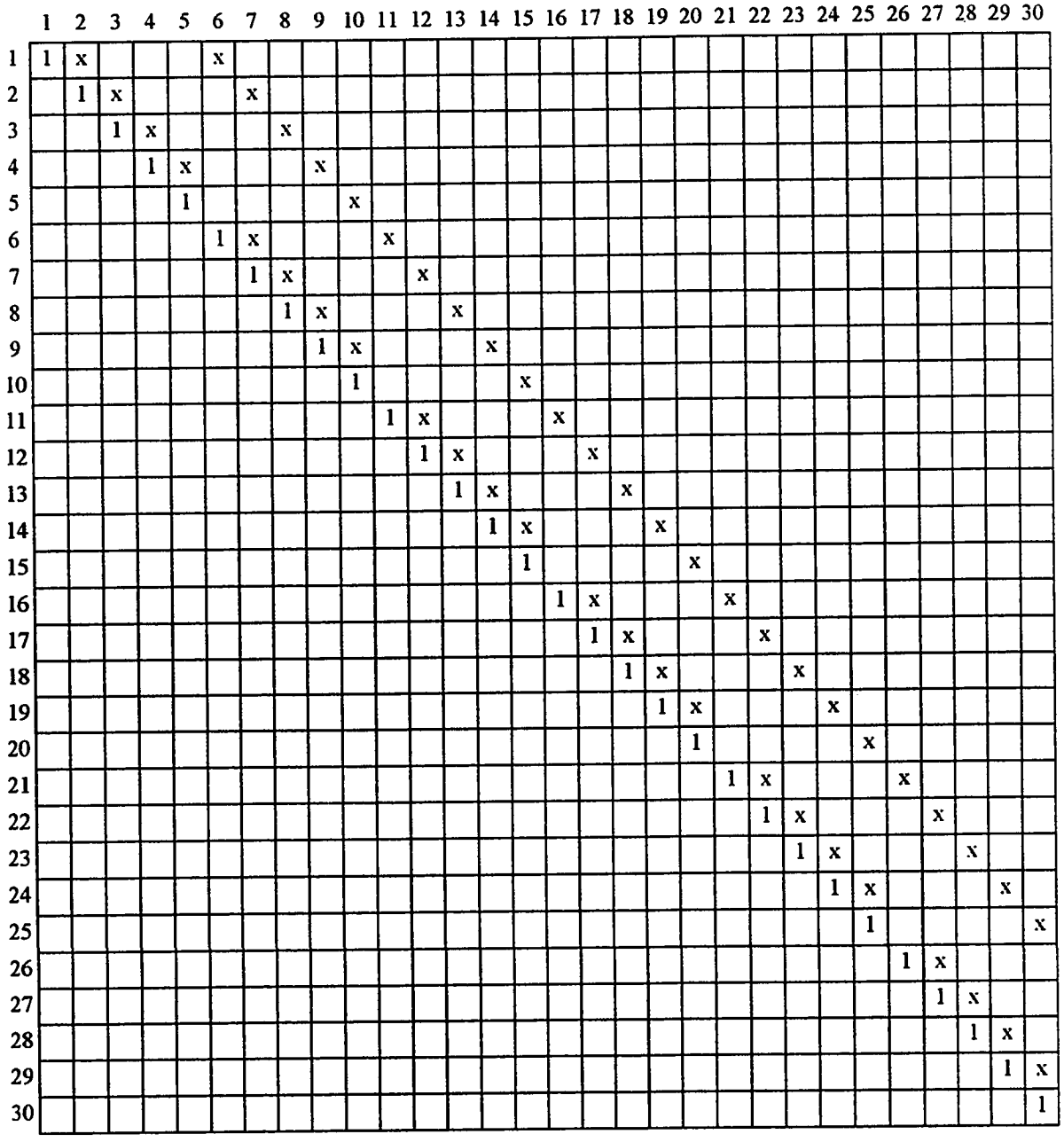


Figure 45 - Form of **U** Matrix for **SIP** Method. Non-zero Entries are Indicated by x. The Diagonal Entries are Equal to 1.

$$e_{ij} = e_{ij}/(1+\alpha c_{i,j-1}) \tag{4.42}$$

$$a_{ij} = a_{ij}/(1+\alpha f_{i,j-1}) \tag{4.43}$$

$$b_{ij} = b_{ij} + \alpha e_{ij} c_{i,j-1} + \alpha a_{ij} f_{i-1,j} - e_{ij} f_{i,j-1} - a_{ij} c_{i-1,j} \tag{4.44}$$

$$c_{ij} = (c_{ij} - \alpha e_{ij} c_{i,j-1})/ b_{ij} \tag{4.45}$$

$$f_{ij} = (f_{ij} - \alpha a_{ij} f_{i-1,j})/ b_{ij} \tag{4.46}$$

where $\alpha$ is an iteration parameter. Using Equation (4.41) above, the iteration procedure can be written as :

$$(\mathbf{A} + \mathbf{N})\mathbf{P}^{(\nu+1)} = (\mathbf{A}+\mathbf{N})\mathbf{P}^{(\nu)} - (\mathbf{A}\mathbf{P}^{(\nu)} - \mathbf{d}) \tag{4.47}$$

Equation (4.47) is written in residual form as

$$\mathbf{R}^{(\nu)} = \mathbf{A}\mathbf{P}^{(\nu)} - \mathbf{d} \tag{4.48}$$

$$\delta^{(\nu+1)} = \mathbf{P}^{(\nu+1)} - \mathbf{P}^{(\nu)} \tag{4.49}$$

and rewrite Equation (4.47) as

$$(\mathbf{A}+\mathbf{N})\delta^{(\nu+1)} = -\mathbf{R}^{(\nu)} \tag{4.50}$$

or

$$\mathbf{L}\,\mathbf{U}\,\delta^{(\nu+1)} = -\mathbf{R}^{(\nu)} \tag{4.51}$$

The elements of **L** and **U** are computed from equations (4.42) to (4.46). The solution is obtained by defining a vector $v$ so that

$$\mathbf{L}\,v = -\mathbf{R}^{(\nu)} \tag{4.52}$$

The elements of vector $v$ may be computed by forward substitution. From equations ( 4.51 ) and (4.52) we see that the following equation results

$$\mathbf{U}\,\delta^{(\nu+1)} = v \tag{4.53}$$

and this equation may be used to compute $\delta^{(\nu+1)}$ by backward substitution.

Stone recommends the use of a sequence of iteration parameters in a cycle. These parameters are geometrically spaced between 0 and $\alpha_{max}$, where

$$(1 - \alpha_{max}) = \text{minimum over the grid of}$$

$$\left[ \frac{\pi^2}{(2I^2)\left[1 + \frac{(\Delta x)^2\ \lambda\ Y}{(\Delta y)^2\ \lambda\ X}\right]} \;,\quad \frac{\pi^2}{(2J^2)\left[1 + \frac{(\Delta y)^2\ \lambda\ X}{(\Delta x)^2\ \lambda\ Y}\right]} \right] \tag{4.54}$$

where $\lambda$ is transmissibility and is equal to $k/(\mu B)$. The iteration parameters are computed from

$$(1 - \alpha_m) = (1 - \alpha_{max})^{m/(M-1)}$$
$$m = 0, 1, 2, \ldots, M - 1 \tag{4.55}$$

where M is the number of parameters per cycle. Stone recommends the use of a minimum of four parameters, each used twice, per cycle.

The elements of **L** and **U** and vector $\nu$ are obtained recursively according to the ordering of unknowns, which is in the order of increasing i for rows j = 1, 2, ... , J. The elements of $\delta$ are then obtained in the reverse order. Thus for odd numbered iterations

$$i = 1, \ 2, \ \ldots, I$$

for each value of j in the order

$$j = 1, \ 2, \ \ldots, J$$

and for even numbered iterations i is allowed to increment as before,

$$i = 1, \ 2, \ \ldots, I$$

but this is done for each value of J in the reverse order

$$j = J, \ J - 1, \ \ldots, 1$$

This reversal in direction is important for convergence of the method in some cases and may be implemented without a separate coding for even numbered iterations [13]. A flowchart of SIP is shown in Figure 46.

## 4.7 Conjugate Gradient-Truncated Direct Method, CGTD

This method solves the reservoir simulation pressure equation using preconditioned conjugate gradients. The preconditioning is based on an approximate inverse to the matrix of pressure equations. The conjugate gradient method applies directly only to symmetric matrices. The reservoir simulation

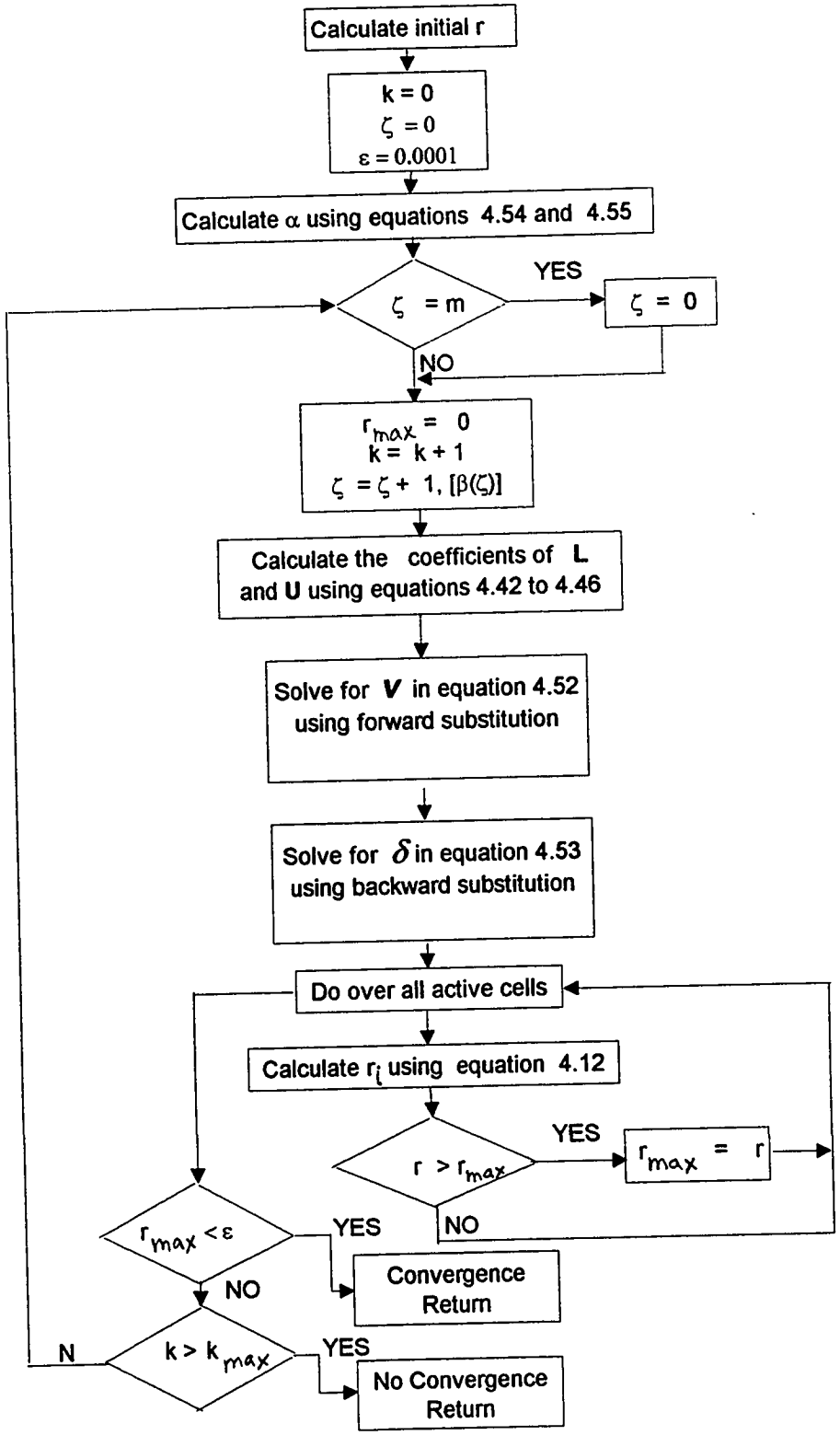



Figure 46 - SIP Flow Chart.

matrix of pressure equations is nonsymmetric. In this method this is overcome by solving a symmetric approximation to the matrix of pressure equations.

This method improves early convergence of the conjugate gradient method through the use of preconditioning, which in this case consisted of premultiplying the linear system by an approximate inverse of the symmetric matrix. The approximate inverse of the equations is generated by direct elimination. Since the matrix of pressure equations is sparse, in direct elimination many of the zeros are filled in by nonzero induced terms. During the elimination process all but a few of the largest of them are ignored or accounted for in an approximate way, and this is how this method was called a truncated direct method [12]. A description of this method follows.

Given the general pressure equation (4.3)

$$\mathbf{A\ P = d}$$

One of the individual equations making up this matrix equation written in two dimensions is as follows

$$\begin{aligned} & e_{ij}P_{i,j-1} + a_{ij}P_{i-1,j} + (e_{ij} + a_{ij} + c_{ij} + f_{ij})P_{ij} \\ & + c_{ij}P_{i+1,j} + f_{ij}P_{i,j+1} + O_{ij}P_{ij} = d_i \end{aligned} \tag{4.56}$$

The $O_{ij}$ term sums the contribution of the compressibility and a pressure-boundary-condition well if present. The flow coefficients $e_{ij}$, $a_{ij}$, $c_{ij}$, and $f_{ij}$ are total fluid transmissibilities. In general

$$e_{ij} \neq f_{ij}$$

$$a_{ij} \neq c_{ij}$$

Hence, the matrix **A** is not symmetric. Therefore, in order to use the conjugate gradient method which applies only to symmetric matrices, a symmetric approximation of the matrix **A** is used instead. The approximate symmetric matrix is formed as follows :

First, the matrix **A** is split into two parts as

$$\mathbf{A} = \mathbf{D} + \mathbf{T} \tag{4.57}$$

The matrix **D** is diagonal and is made up of the terms caused by compressibility and pressure-boundary-condition wells. The matrix **T** is made up of the flow coefficients and is nonsymmetric. A symmetric approximation to the matrix **T** , which is termed as $\mathbf{T}'$ is constructed as follows

$$a'_{ij} = c'_{i-1,j} = \frac{(a_{ij} + c_{i-1,j})}{2} \tag{4.58}$$

$$e'_{ij} = f'_{i,j-1} = \frac{(e_{ij} + f_{i,j-1})}{2} \tag{4.59}$$

The main diagonal of $\mathbf{T}'$ is the sum of the off-diagonals:

$$a'_{ij} + c'_{ij} + e'_{ij} + f'_{ij}$$

The symmetric approximation of the matrix **A** is formed as

$$\mathbf{A}_S = \mathbf{D} + \mathbf{T}' \tag{4.60}$$

It cannot be assumed that the solution to $\mathbf{A}_S \mathbf{P} = \mathbf{d}$ will yield pressures accurate enough for the purpose of reservoir simulation. Therefore an iterative solution is used as follows:

$$\mathbf{A}_S \Delta \mathbf{P}_n = \mathbf{r}^n \tag{4.61}$$

where

$$\mathbf{r}^n = \mathbf{d} - \mathbf{A}\mathbf{P}^n \tag{4.62}$$

and

$$\Delta \mathbf{P}^n = \mathbf{P}^{n+1} - \mathbf{P}^n$$

Equation (4.61) is solved using the conjugate gradient method iteratively untill the desired convergence criteria is met. The conjugate gradient method is used in the same way that iterative methods are used. The solution proceeds step by step until the errors are reduced to some tolerance. In theory, the method converges to the exact solution but requires excessive number of steps, which could be very large in case of a large number of equations. However, with the use of an approximate inverse of the matrix to be solved, the method is preconditioned and thus early convergence is accelerated. The iteration performed using the conjugate gradient method to solve equation (4.61) as described by Watts [11] is as follows.

In the equations below m denotes the inner iteration and n the outer iteration. Each outer iteration is a solution of $\mathbf{A}_S \Delta \mathbf{P}^n = \mathbf{r}^n$ for $\Delta \mathbf{P}^n$. Each inner iteration is an iteration of the conjugate gradient method. At each of these inner iterations, a new estimate of $\Delta \mathbf{P}^n$ is computed by

$$\Delta \mathbf{P}^{m+1,n} = \Delta \mathbf{P}^{m,n} + \alpha_m \mathbf{s}_m \tag{4.63}$$

where

$$\mathbf{s}_m = \mathbf{H}^{-1} \mathbf{r}_{m,n} + \beta_m \mathbf{s}_{m-1} \tag{4.64}$$

The two scalars $\alpha_m$ and $\beta_m$ are defined as follows :

$$\alpha_m = \frac{(\mathbf{r}_{(m,n)})^T \mathbf{H}^{-1} \mathbf{r}_{(m,n)}}{(\mathbf{s}_m)^T \mathbf{A}_S \mathbf{s}_m} \tag{4.65}$$

$$\beta_m = \frac{(\mathbf{r}_{(m,n)})^T \mathbf{H}^{-1} \mathbf{r}_{(m,n)}}{(\mathbf{r}_{(m-1,n)})^T \mathbf{H}^{-1} \mathbf{r}_{(m-1,n)}} \tag{4.66}$$

with

$$\beta_1 = 0 \tag{4.67}$$

The residual of the inner iteration is obtained by

$$\mathbf{r}_{m,n} = \mathbf{r}_{0,n} - \mathbf{A}_S \Delta \mathbf{P}^{m,n} \tag{4.68}$$

where

$$\Delta \mathbf{P}^{m,n} = \mathbf{P}^{m,n} - \mathbf{P}^{0,n} \tag{4.69}$$

The matrix is assumed to be approximately equal to the matrix $\mathbf{A}_S$. Hence $\mathbf{H}^{-1}$ is an approximate inverse of $\mathbf{A}_S$. This approximate inverse is constructed using the truncated direct elimination procedure. In two-dimensional problems it was found best to retain two additional induced terms directly inside each of the flanking diagonals and three additional terms on each side of the main diagonal. Moreover, a diagonal ordering of the grid cells is recommended for anisotropic

problems with transmissibility much higher in one dimension than the other. A flow chart of this method is shown in Figure 47.

## 4.8 Nested Factorization, NF

This method comprises a novel preconditioning for the Conjugate Gradient and the Orthomin iteration procedures. It differs from the incomplete Cholesky factorization in that it does not form the preconditioning matrix from strictly upper and lower factors. Instead, it constructs block lower and upper factors using a procedure which adds one dimension at a time to the preconditioning matrix [12].

Nested Factorization produces a mass conserving approximation to the coefficient matrix [12]. In most reservoir simulators, the sum of the elements on the right hand of the linear equation $\mathbf{A}\ \mathbf{P} = \mathbf{d}$ is equal to the net rate of mass accumulation in the reservoir. If an initial approximate solution, $\mathbf{P}_0$ to this equation is obtained by solving :

$$\mathbf{B}\ \mathbf{P}_0 = \mathbf{d} \tag{4.70}$$

then the error in the net rate of mass accumulation ( the material balance error ) is given by the sum of the elements in the initial residual, $\mathbf{r}_0$,

$$\mathbf{r}_0 = \mathbf{d} - \mathbf{A}\ \mathbf{P}_0$$

$$= (\ \mathbf{B} - \mathbf{A}\ )\ \mathbf{P}_0 \tag{4.71}$$

This sum is zero if $\mathbf{B}$ is chosen in such a way that

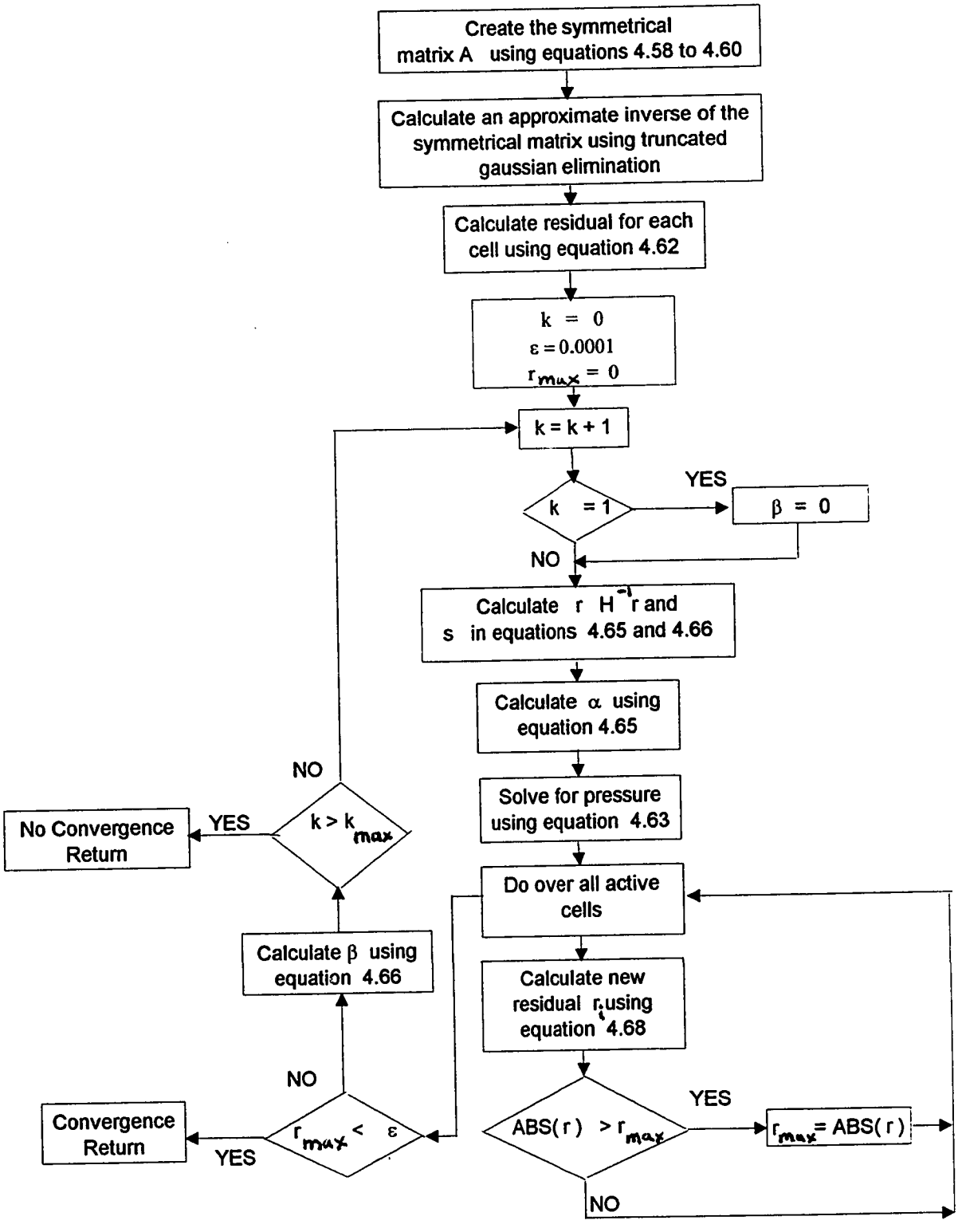



Figure 47 - CGTD Method Flow Chart

$$\text{colsum}(\mathbf{B}) = \text{colsum}(\mathbf{A}) \tag{4.72}$$

where colsum( **A** ) is the diagonal matrix formed by summing the elements of **A** in columns. Equation (4.72) ensures that the sum of the elements of **A .v** is equal to the sum of **B.v** for any vector **v**. The following describes how the preconditioning matrix **B** is formed using this method [12 ].

The formation of the finite difference equations for the two-dimensional 5 x 4 grid shown in Figure 48 will result in a banded matrix of the form

$$\mathbf{A} = \mathbf{D} + \mathbf{L}_1 + \mathbf{U}_1 + \mathbf{L}_2 + \mathbf{U}_2 \tag{4.73}$$

where **D** is the main diagonal, $\mathbf{L}_1$ and $\mathbf{L}_2$ are the lower bands and $\mathbf{U}_1$ , $\mathbf{U}_2$ are the upper bands, Figure 49. Because of gaps in $\mathbf{L}_1$ and $\mathbf{U}_1$ corresponding to the edge of the reservoir, **A** may be regarded as a block tridiagonal matrix. From this perspective, the upper and lower bands ( $\mathbf{U}_2$ and $\mathbf{L}_2$ ) contain interactions between lines, and the block diagonal matrix elements contain interactions within lines. On a yet smaller scale, the matrix of interactions within a line is itself tridiagonal. The nested tridiagonal structure of the matrix **A** is exploited in this method by defining the preconditioning matrix **B** in this sequence

$$\mathbf{B} = ( \mathbf{T} + \mathbf{L}_2 ) . ( \mathbf{I} + \mathbf{T}^{-1} . \mathbf{U}_2 ) \tag{4.74}$$

$$\mathbf{T} = ( \mathbf{M} + \mathbf{L}_1 ) . ( \mathbf{I} + \mathbf{M}^{-1} . \mathbf{U}_1 ) \tag{4.75}$$

where **M** is a diagonal matrix. **M** is defined by

| | I=1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| J=1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 6 | 7 | 8 | 9 | 10 |
| 3 | 11 | 12 | 13 | 14 | 15 |
| 4 | 16 | 17 | 18 | 19 | 20 |

Figure 48 - Standard Ordering Applied to
a 5 x 4 2-D Grid

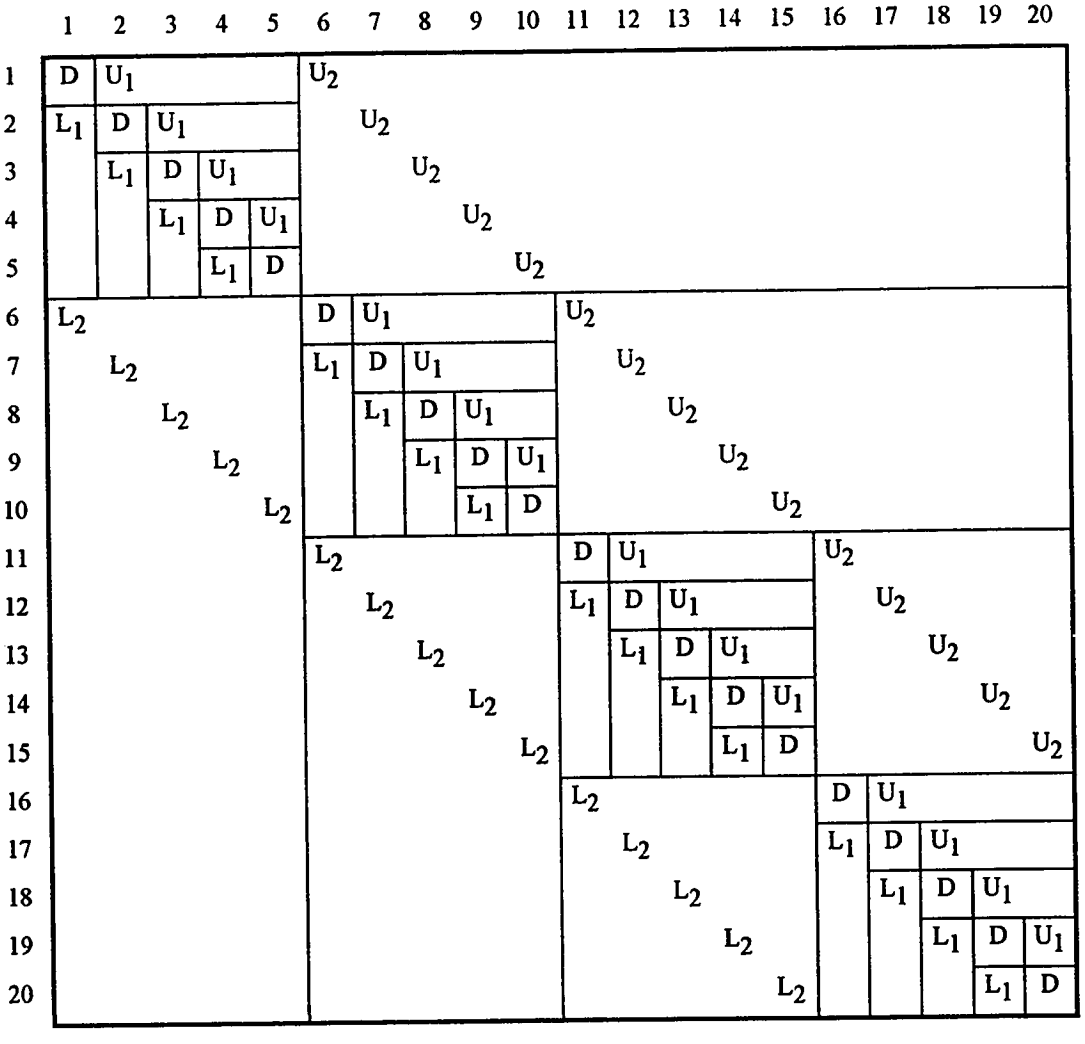



Figure 49 - The Nested Structure of the 20 x 20 Coefficient Matrix for the 5 x4 Grid in Figure - 48, Using Standard Ordering.

$$\mathbf{M} = \mathbf{D} - \mathbf{L}_1 . \mathbf{M}^{-1} . \mathbf{U}_1 - \text{colsum}( \mathbf{L}_2 . \mathbf{T}^{-1} . \mathbf{U}_2 ) \quad (4.76)$$

then

$$\mathbf{B} = \mathbf{A} - \mathbf{L}_2 . \mathbf{T}^{-1} . \mathbf{U}_2 - \text{colsum}( \mathbf{L}_2 . \mathbf{T}^{-1} . \mathbf{U}_2 ) \quad (4.77)$$

and the colsum of the error matrix, $\mathbf{B} - \mathbf{A}$ is zero.

The solution using this method is hierarchical. The procedure is to compute an approximate solution to the residual equation

$$\mathbf{B} \ \Delta \mathbf{P}_m = \mathbf{r}_m \quad (4.78)$$

where

$$\mathbf{r}_m = \mathbf{d} - \mathbf{A} \ \mathbf{P}_m \quad (4.79)$$

and $\mathbf{r}_m$ is the residual after m iterations, and **B** is an approximation to **A** chosen so that the solution may be obtained efficiently. The search direction $\Delta \mathbf{P}_m$, is used to update the solution, using

$$\mathbf{P}_{m+1} = \mathbf{P}_m + \Delta \mathbf{P}_m \quad (4.80)$$

Thus at the outer most level the following equation is solved

$$( \mathbf{T} + \mathbf{L}_2 ) . ( \mathbf{I} + \mathbf{T}^{-1} . \mathbf{U}_2 ) . \Delta \mathbf{P} = \mathbf{r} \quad (4.81)$$

using

$$\Delta \mathbf{P} = \mathbf{T}^{-1} . ( \mathbf{r} - \mathbf{L}_2 . \Delta \mathbf{P} ) \quad (4.82)$$

and

$$\Delta \mathbf{P} = \Delta \mathbf{P} - \mathbf{T}^{-1} . \mathbf{U}_2 . \Delta \mathbf{P} \quad (4.83)$$

Equation (4.82) is solved for $\Delta P$ one line at a time, starting with the first line and progressing forward until $\Delta P$ is known on each line. Similar considerations apply to the solution of equation (4.83) which is performed in the reverse order starting with the last line and sweeping backwards one line at a time.

During the solution of equations (4.82) and (4.83) vectors of the type $w = T^{-1} . v$ within each line must be computed. This involves the solution of the tridiagonal equation

$$( M + L_1 ) . ( I + M^{-1} . U_1 ) . w = v \tag{4.84}$$

using

$$w = M^{-1} . ( v - L_1 . w ) \tag{4.85}$$

and

$$w = w - M^{-1} . U_1 . w \tag{4.86}$$

The above equations are solved by sweeping first forwards, then backwards through the cells in the line.

Before iteration begins the diagonal matrices $M^{-1}$ and $M^{-1}.U_1$ must be computed, using

$$M = D - L_1 . M^{-1} . U_1 - \text{colsum} ( L_2 . T^{-1} . U_2 ) \tag{4.87}$$

The calculation proceeds a cell at a time; thus when $M^{-1}$ is known for a cell, the contribution $L_1 . M^{-1} . U_1$ to the value of M in the next cell can be computed. When $M^{-1}$ is known for an entire line, the contribution colsum( $L_2 . T^{-1} . U_2$ ) to the value of M on the next line is calculated.

The work count requirement for this method for a 2 phase 2 D problem is

56 + 62 m ( floating point multiplications and divisions per cell ) where m is the number of iterations. A flow chart of this method is shown in Figure 50.

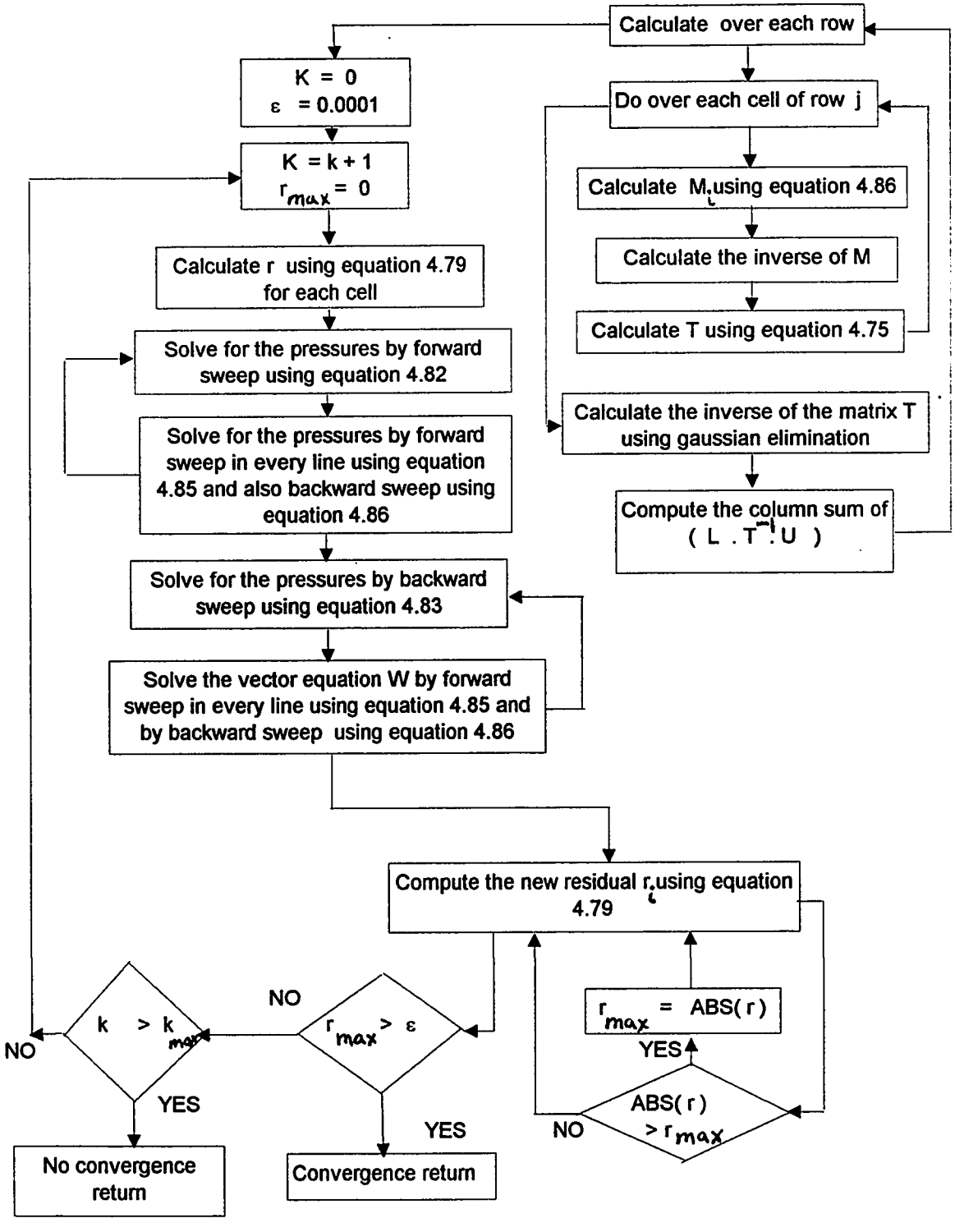



Figure 50 - NF Flow Chart

# CHAPTER 5

# THE RESERVOIR PROBLEM

The methods discussed previously will be tested in solving a given reservoir problem.

The reservoir problem used is a two-phase, oil-water, two-dimensional reservoir water flooding simulation problem. Comparisons will be made of the computing efficiency in terms of simulation time as well as the memory space requirement for each method.

The test problem chosen for this study is to simulate a five-spot water flooding in a large hydrocarbon reservoir. The heterogeneity of this reservoir was found not to vary drastically between wells which have been spaced one kilometer apart. Thus, a homogenous one layer model can be used to model this reservoir for the purpose of this study.

A quadrant of the five spot pattern was modeled with a 19 x 19 uniform grid. The number of grids was 361 and was found to be sufficient to distinguish between the power of the iterative and the direct methods. Two wells were included, a producer in grid I = 19 and J = 19, and an injector in grid I = 1 and J = 1. The injection well was assigned a maximum injection rate of 10000 STB/D with the injection pressure not to exceed 4000 psig. An injectivity

index of 40 STB/D/psi was used. The producer was assigned a maximum total rate of 10000 STB/D. A productivity index of 40 STB/D/psi was also used. The producer was set at a constant flowing bottom hole pressure of 2500 psig. The rest of the reservoir parameters are listed in Table 16. The PVT data and the relative permeability and capillary pressure data are are listed in Tables 17 and 18.

A simple two-phase three dimensional reservoir simulator [23] was used during this study. The solution techniques were used to solve for pressures implicitly, while saturations were calculated explicitly. The results of the computations are discussed in the following two chapters.

TABLE - 16 Reservoir Parameters of the Test Problem

| | |
|---|---|
| Number of Grids in the x Direction = | 19 |
| Number of Grids in the y Direction = | 19 |
| Number of Grids in the z Direction = | 1 |
| Initial Water Saturation, ( fraction ) = | 0.10 |
| Initial Pressure at the Oil-Water Contact, ( psig ) = | 3075 |
| Depth of the Oil-Water Contact , ( ft ) = | 6500 |
| Reference Depth, ( ft ) = | 5300 |
| Oil Density, ( lb / ft ) = | 47.95 |
| Water Density, ( lb / ft ) = | 62.35 |
| Rock Compressibility, ( $psi^{-1}$ ) = | $3 \times 10^{-6}$ |
| Length of Each Grid in the x-direction, ( ft ) = | 250 |
| Length of Each Grid in the y-direction, ( ft) = | 250 |
| Reservoir Thickness, ( ft) = | 100 |
| Porosity, ( fraction ) = | 0.20 |
| Permeability in the x-direction, ( md ) = | 500 |
| Permeability in the y-direction, ( md ) = | 500 |
| Permeability in the z-direction, ( md ) = | 500 |
| Depth at Top of Reservoir, ( ft ) = | 5300 |

TABLE 17 - PVT Data

| PRESSURE ( psig) | OIL VISCOSITY ( cp) | OIL FORMATION VOLUME FACTOR ( res bbl / STB ) | WATER VISCOSITY ( cp ) | WATER FORMATION VOLUME FACTOR ( res bbl / STB ) |
|---|---|---|---|---|
| 2014.7 | 0.695 | 1.435 | 0.31 | 1.035 |
| 3014.7 | 0.757 | 1.420 | 0.31 | 1.031 |
| 4014.7 | 0.819 | 1.406 | 0.31 | 1.028 |
| 6014.7 | 0.944 | 1.376 | 0.31 | 1.022 |
| 9014.7 | 1.130 | 1.332 | 0.31 | 1.013 |

TABLE 18 - Relative Permeability and Capillary Pressure Data

| $S_w$ ( fraction ) | $k_{ro}$ | $k_{rw}$ | CAPILLARY PRESSURE ( psig ) |
|---|---|---|---|
| 0.0 | 1.0 | 0.0 | 25.0 |
| 0.08 | 1.0 | 0.0 | 25.0 |
| 0.10 | 0.83 | 0.003 | 20.0 |
| 0.15 | 0.60 | 0.013 | 2.0 |
| 0.20 | 0.46 | 0.025 | -1.0 |
| 0.25 | 0.35 | 0.04 | -2.1 |
| 0.35 | 0.175 | 0.065 | -2.3 |
| 0.40 | 0.120 | 0.100 | -2.4 |
| 0.45 | 0.075 | 0.130 | -2.5 |
| 0.50 | 0.040 | 0.160 | -2.7 |
| 0.55 | 0.015 | 0.200 | -2.8 |
| 0.65 | 0.005 | 0.280 | -2.9 |
| 0.70 | 0.001 | 0.325 | -3.0 |
| 0.75 | 0.0 | 0.390 | -3.1 |

# CHAPTER 6

# COMPARATIVE EVALUATION

The reservoir problem was solved using the following methods :

A- Direct methods :

1 - **Gaussian Elimination** using standard ordering,
( **STANDARD - GAUSS** )

2 - **Gaussian Elimination** using **D4** ordering, ( **D4 - GAUSS** )

3 - **Matrix Decomposition** using standard ordering, ( **STBAND** )

4 - **RAD** Method

B- Iterative methods :

1 - **PSOR**

2 - **LSOR**

3 - **ADIP**

4 - **SIP**

5 - **CGTD**

6 - **NF**

The problem was simulated for only 40 days as shown in Table 19. Since the objective of the thesis is to compare the time and storage requirements, the simulation was carried for a short time just to obtain reasonable estimates of the

cpu time per step used by each method. The results of the simulation are listed in Tables 20A and 20B, and are plotted in Figures 51 - 53. The analysis of the results for each method are discussed in the next sections.

TABLE 19 - Simulation Time and Constraints

| Time ( days ) | Maximum Change in Saturation (fraction) | Maximum Change in Pressure ( psi ) | Maximum Time Step Size ( days ) |
|---|---|---|---|
| 0.10 | 0.05 | 200 | 2.00 |
| 0.15 | 0.05 | 200 | 2.00 |
| 0.25 | 0.05 | 200 | 2.00 |
| 0.50 | 0.05 | 200 | 2.00 |
| 1.00 | 0.05 | 200 | 2.00 |
| 3.00 | 0.05 | 200 | 2.00 |
| 5.00 | 0.05 | 200 | 2.00 |
| 10.00 | 0.05 | 200 | 2.00 |
| 15.00 | 0.05 | 200 | 2.00 |
| 20.00 | 0.05 | 200 | 2.00 |
| 25.00 | 0.05 | 200 | 2.00 |
| 30.00 | 0.05 | 200 | 2.00 |
| 35.00 | 0.05 | 200 | 2.00 |
| 40.00 | 0.05 | 200 | 2.00 |

**TABLE 20A - Injection and Production Volumes.**

| TIME ( DAYS ) | CUMULATIVE WATER INJECTED ( STB ) | CUMULATIVE OIL PRODUCED ( STB ) | CUMULATIVE WATER PRODUCED ( STB ) | CUMULATIVE WATER OIL RATIO |
|---|---|---|---|---|
| 0.00 | 0 | 0 | 0.0 | 0.000 |
| 0.10 | 1000 | 682 | 8.1 | 0.012 |
| 0.25 | 2500 | 1550 | 18.5 | 0.012 |
| 0.50 | 5000 | 2859 | 34.1 | 0.012 |
| 1.00 | 10000 | 5271 | 63.0 | 0.012 |
| 2.00 | 20000 | 9785 | 116.9 | 0.012 |
| 5.00 | 50000 | 22810 | 272.5 | 0.012 |
| 10.00 | 100000 | 44900 | 536.5 | 0.012 |
| 15.00 | 150000 | 68220 | 815.3 | 0.012 |
| 20.00 | 200000 | 92670 | 1108.0 | 0.012 |
| 25.00 | 250000 | 118200 | 1413.0 | 0.012 |
| 30.00 | 300000 | 144600 | 1729.0 | 0.012 |
| 35.00 | 350000 | 171900 | 2056.0 | 0.012 |
| 40.00 | 400000 | 200000 | 2392.0 | 0.012 |

**TABLE 20B - Injection and Production Rates and Pressures.**

| TIME ( DAYS ) | OIL PRODUCTION RATE ( STB / DAY ) | WATER PRODUCTION RATE ( STB / DAY ) | WATER INJECTION RATE ( STB / DAY ) | INJECTION CELL PRESSURE ( PSI ) | PRODUCTION CELL PRESSURE ( PSI ) |
|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0 | 2805.2 | 2805.20 |
| 0.10 | 6823.66 | 81.08 | 10000 | 2910.3 | 2716.21 |
| 0.25 | 5782.72 | 69.12 | 10000 | 2965.1 | 2682.90 |
| 0.50 | 5235.23 | 62.58 | 10000 | 3007.0 | 2665.39 |
| 1.00 | 4825.05 | 57.66 | 10000 | 3049.5 | 2652.29 |
| 2.00 | 4514.06 | 53.93 | 10000 | 3097.9 | 2642.38 |
| 5.00 | 4317.69 | 51.59 | 10000 | 3201.1 | 2636.13 |
| 10.00 | 4500.13 | 53.79 | 10000 | 3268.7 | 2641.94 |
| 15.00 | 4738.46 | 56.65 | 10000 | 3288.5 | 2649.54 |
| 20.00 | 4960.83 | 59.32 | 10000 | 3314.7 | 2656.63 |
| 25.00 | 5162.87 | 61.76 | 10000 | 3342.0 | 2663.09 |
| 30.00 | 5347.00 | 63.97 | 10000 | 3350.4 | 2668.97 |
| 35.00 | 5514.97 | 66.00 | 10000 | 3362.3 | 2674.35 |
| 40.00 | 5667.43 | 67.83 | 10000 | 3376.3 | 2679.23 |

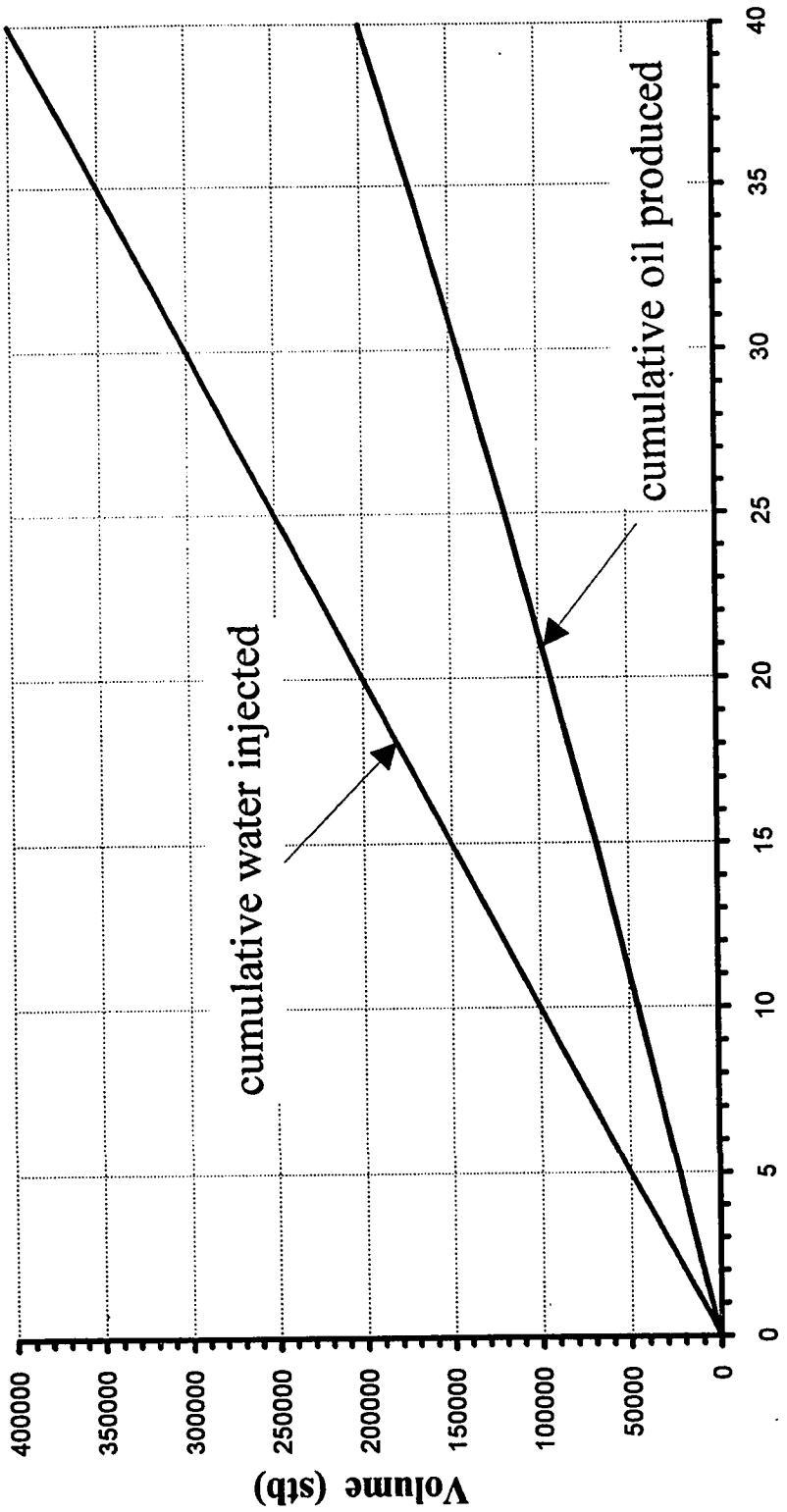



Figure 51 - Cumulative Oil and Water Volumes

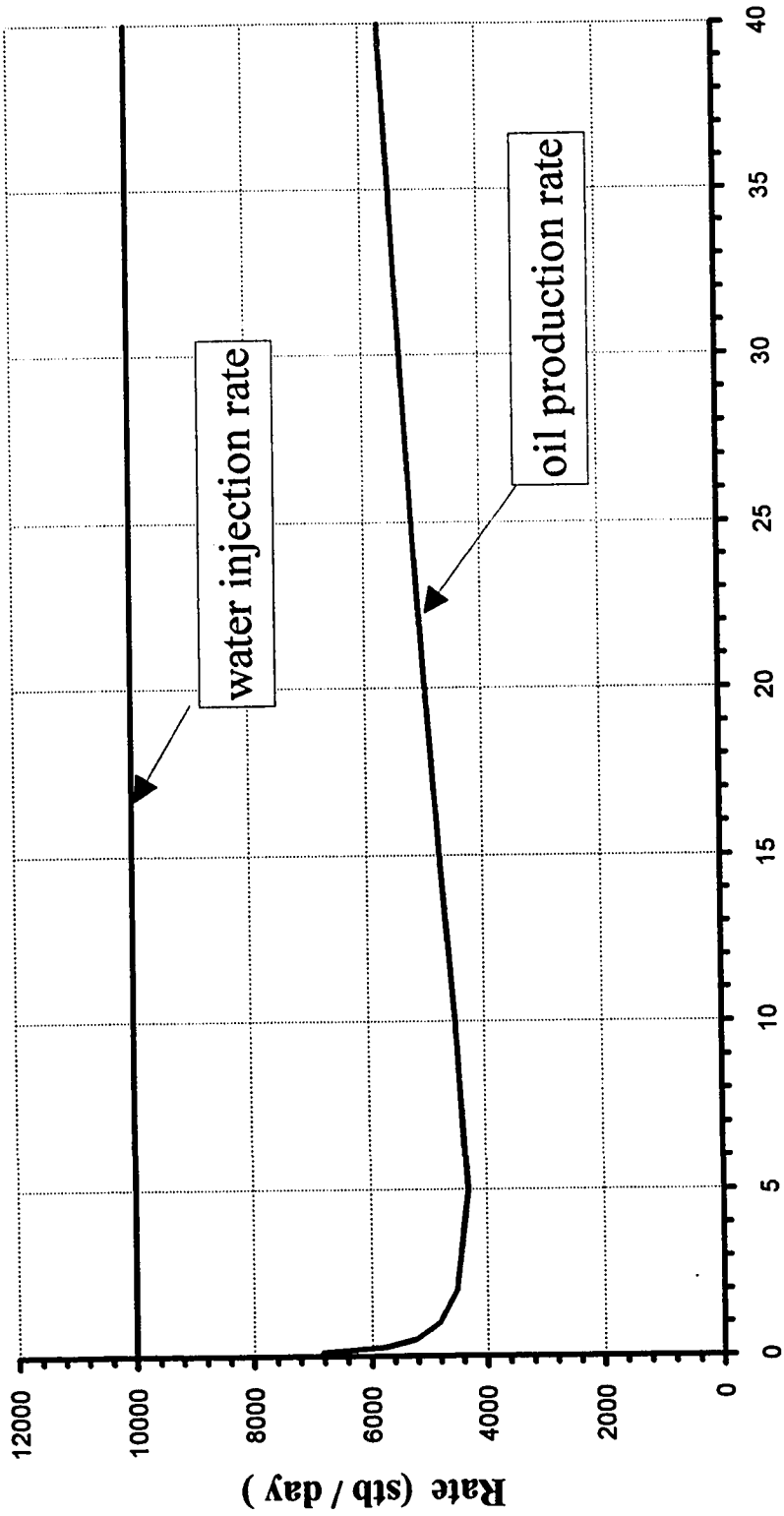



Figure 52 - Oil Production and Water Injection Rates

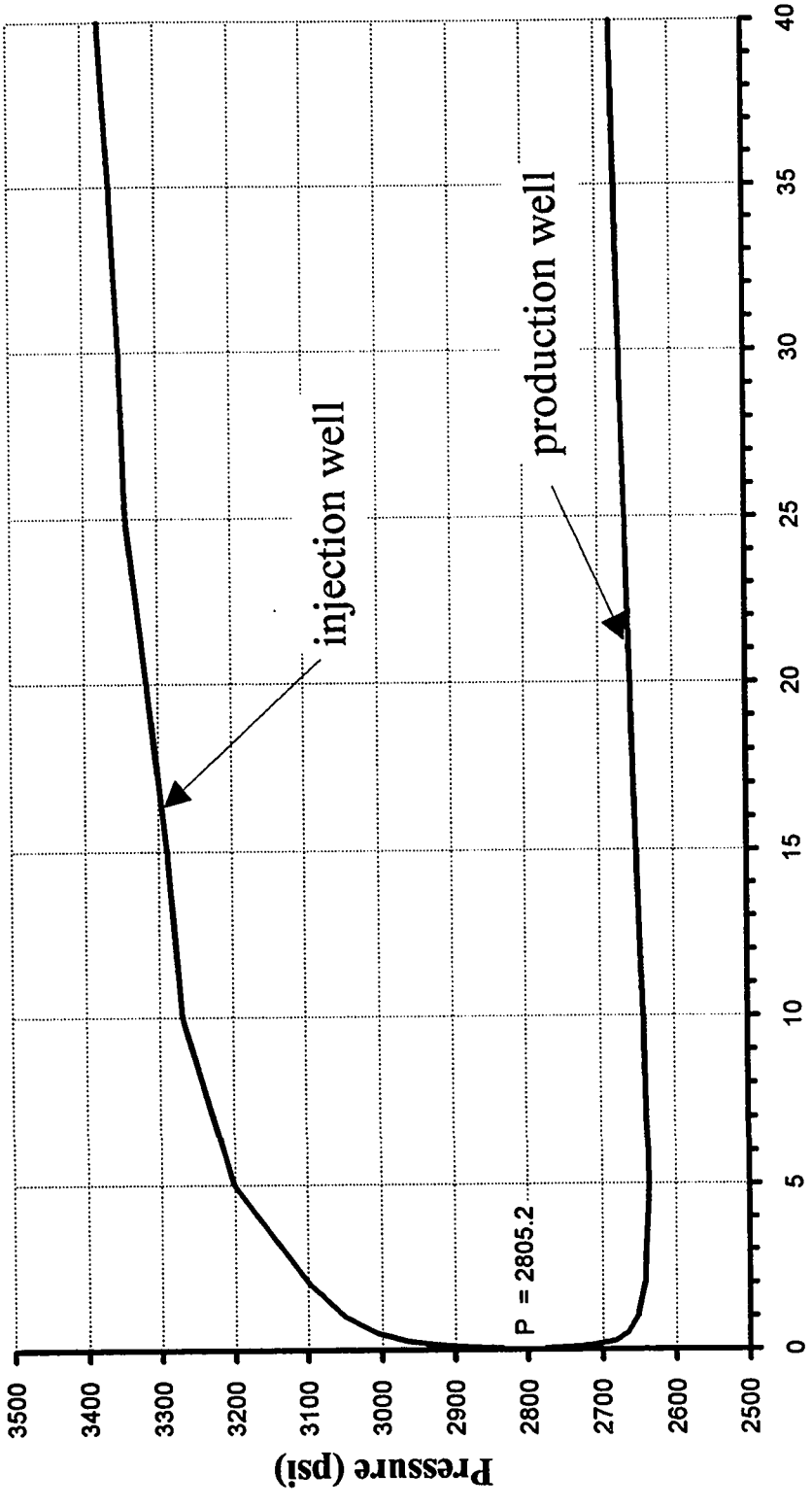



Figure 53 - Injection and Production Cell Pressures

## 6.1 Gaussian Elimination Using Standard Ordering, ( STANDARD - GAUSS )

This is a direct solution method in which the simulation grid cells are ordered using the natural ordering. The pressures are solved for using the process of gaussian elimination. The matrix is first reduced to an upper triangular matrix. Then the pressures are solved for using backward substitution. A listing of the computer program for this method is attached in the appendix.

The time steps and the relevant cpu time used for this method are listed in Table 21. The total cpu time used was 19.45 seconds, resulting in an average of 0.65 seconds per time step. Since this method is a direct solution method the cpu time for individual time steps should be constant. The cpu time per step and the cumulative cpu time used are plotted in Figure 54. This method will always converge to the right solution after the completion of a certain number of calculations. The memory space requirement for this method was 117024 bytes, Table 37.

## 6.2 Gaussian Elimination Using D4 Ordering, ( D4 - GAUSS )

This method is similar to the method discussed above except that the grids are ordered differently. As has been discussed before, the alternating diagonal ordering ( **D4** ) has been found to be one of the most optimum ordering in reservoir simulation to date. The re-ordering of the grid system is performed only once at the beginning of the simulation. The pressures are solved for using the process of gaussian elimination in which the matrix is first reduced to an upper matrix, then the pressures are solved for using backward substitution. A listing of

**TABLE 21 - Simulation Time Steps and cpu Time for STANDARD - GAUSS Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [days] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.650 | 0.65 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.620 | 1.27 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.650 | 1.92 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.630 | 2.55 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.670 | 3.22 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.640 | 3.86 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.660 | 4.52 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.660 | 5.18 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.640 | 5.82 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.650 | 6.47 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.640 | 7.11 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.640 | 7.75 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.670 | 8.42 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.660 | 9.08 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.670 | 9.75 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.640 | 10.39 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.630 | 11.02 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.670 | 11.69 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.620 | 12.31 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.640 | 12.95 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.650 | 13.60 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.680 | 14.28 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.660 | 14.94 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.640 | 15.58 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.650 | 16.23 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.650 | 16.88 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.640 | 17.52 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.630 | 18.15 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.670 | 18.82 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.630 | 19.45 |

**AVERAGE CPU TIME / STEP = 19.45 / 30 = 0.65 seconds**

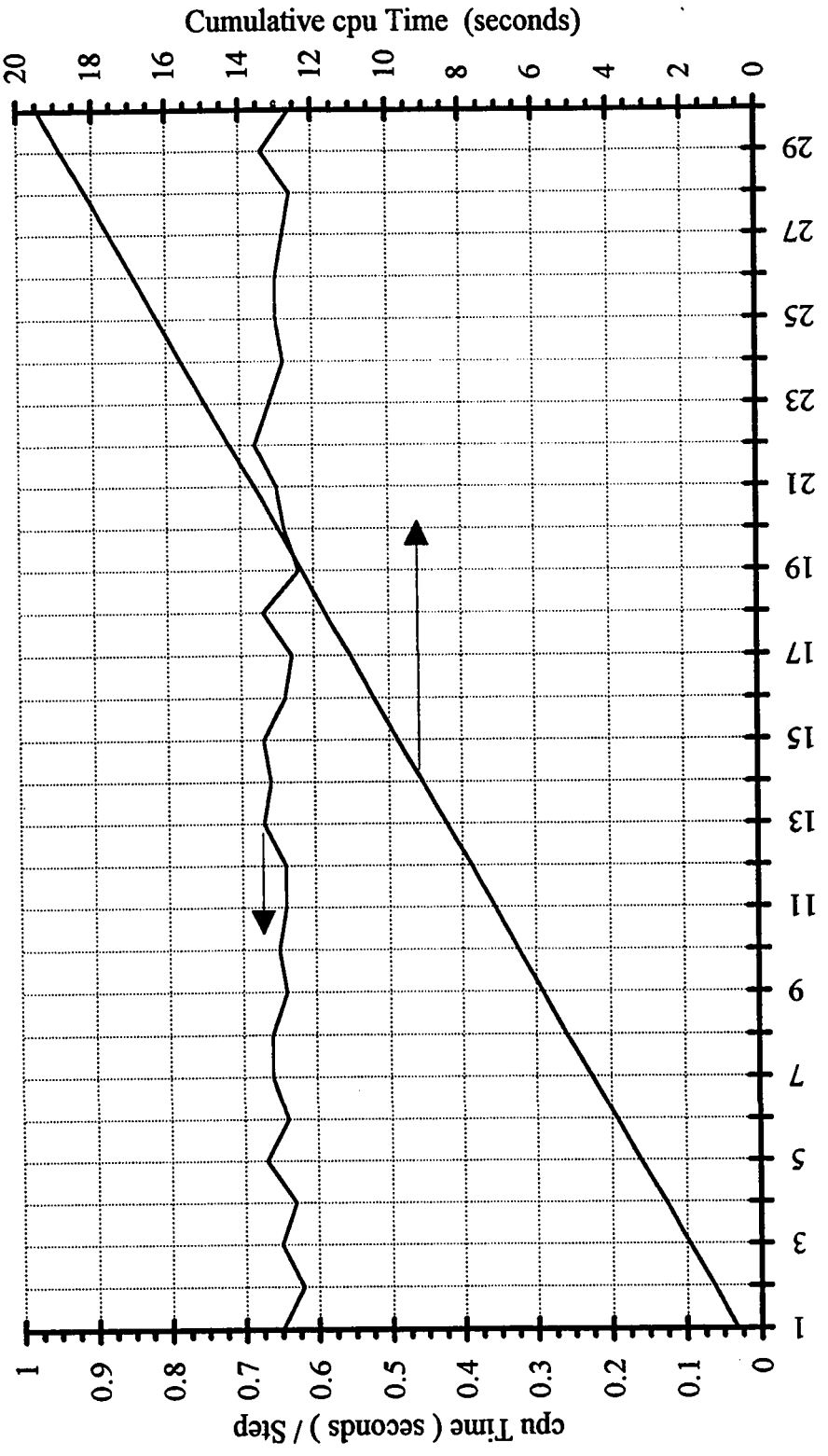



Figure 54 - cpu Time Used by Standard - Gauss Method

the computer program for this method is attached in the appendix.

The time steps and the relevant cpu time consumption for this method are listed in Table 22. The total cpu time used by the method was 9.85 seconds, resulting in an average of 0.33 seconds per time step. This method proved to be the most powerful of the direct methods tested in terms of both cpu time and storage requirements. The cpu time per step and the cumulative cpu time used are plotted in Figure 55. Since this method is a direct method it will always converge to the right solution after the completion of a certain number of calculations. The memory space requirement was 105044 bytes, Table 37.

## 6.3 L / U Decomposition Using Standard Ordering, ( STBAND )

This is a direct solution method in which the simulation grid cells are ordered using the natural ordering [10]. The pressures are solved for using the process of **matrix decomposition** forming a lower matrix **L** and an upper matrix **U**. The matrix is first reduced to an upper matrix. Then the pressures are solved for using forward reduction and then backward substitution. The computer program for this method is listed in the appendix.

The time steps and the relevant cpu time consumption for this method are listed in Table 23. The total cpu time used by this method was 29.74 seconds, resulting in an average of 0.99 seconds per time step, which is higher than that used by the Standard-Gauss which makes use of Gaussian Elimination procedure. This method was found to be the least efficient of the direct methods in terms of both cpu time and memory space requirements. Again, this method is a direct solution method, thus the cpu time for individual time steps should be constant and will always converge to the right solution after the completion of a certain number of calculations. The cpu time per step and the cumulative cpu time used are plotted

**TABLE 22 - Simulation Time Steps and cpu Time for D4 - GAUSS Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.420 | 0.42 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.330 | 0.75 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.320 | 1.07 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.340 | 1.41 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.330 | 1.74 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.310 | 2.05 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.330 | 2.38 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.320 | 2.70 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.340 | 3.04 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.330 | 3.37 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.310 | 3.68 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.310 | 3.99 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.330 | 4.32 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.330 | 4.65 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.320 | 4.97 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.330 | 5.30 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.350 | 5.65 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.330 | 5.98 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.320 | 6.30 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.320 | 6.62 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.320 | 6.94 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.330 | 7.27 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.330 | 7.60 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.310 | 7.91 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.310 | 8.22 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.340 | 8.56 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.330 | 8.89 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.320 | 9.21 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.320 | 9.53 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.320 | 9.85 |

**AVERAGE CPU TIME / STEP = 9.85 / 30 = 0.33 seconds**

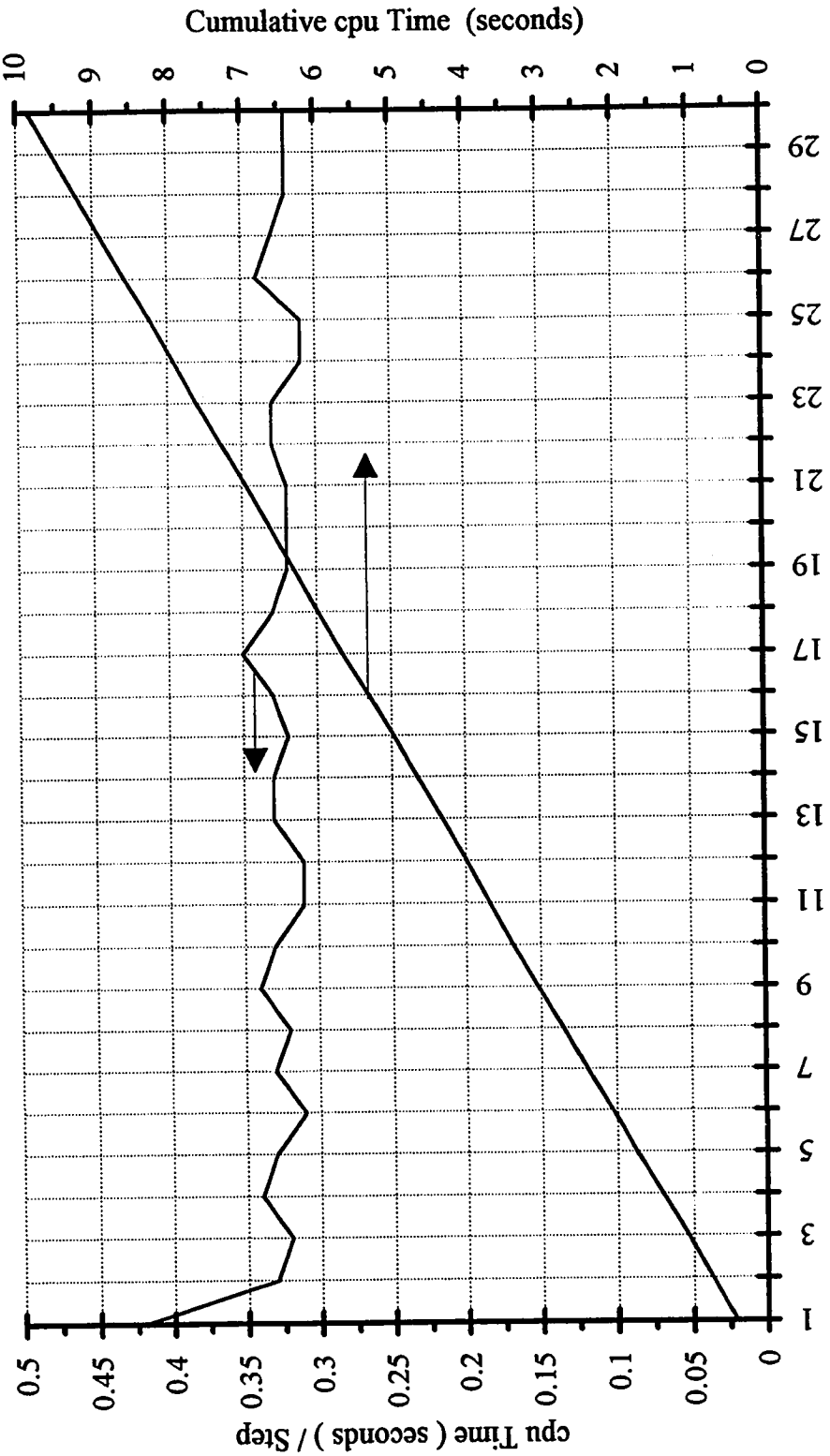



Figure 55 - cpu Time Used by D4 - Gauss Method

**TABLE 23 - Simulation Time Steps and cpu Time for STBAND Method.**

| TIME STEP NUMBER | TIME [ days] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 1.020 | 1.02 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 1.010 | 2.03 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.970 | 3.00 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 1.010 | 4.01 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 1.010 | 5.02 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.960 | 5.98 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 1.000 | 6.98 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.990 | 7.97 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 1.010 | 8.98 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 1.010 | 9.99 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.950 | 10.94 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.990 | 11.93 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.990 | 12.92 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.960 | 13.88 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.990 | 14.87 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.980 | 15.85 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.990 | 16.84 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.990 | 17.83 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 1.000 | 18.83 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.990 | 19.82 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.990 | 20.81 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.990 | 21.80 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 1.000 | 22.80 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 1.010 | 23.81 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 1.030 | 24.84 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 1.000 | 25.84 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.980 | 26.82 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.970 | 27.79 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.960 | 28.75 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.990 | 29.74 |

**AVERAGE CPU TIME / STEP = 29.74 / 30 = 0.99 seconds**

in Figure 56. The memory space requirement was 180472 bytes, Table 37.

## 6.4 RAD Method

In this method the simulation grid cells are reordered differently using the restricted alternating diagonal ordering discussed in section 3.6. The reordering is only performed once at the beginning of the simulation. The coding of this method is based on the direct method of matrix decomposition. Since all the elements positions are known before hand, this method is a straight forward solution procedure in which computations are primarily performed on the lower right hand quadrant of the matrix, which makes it an efficient solver. A listing of the computer program for this method is attached in the appendix.

The time steps and the relevant cpu time consumption for this method are listed in Table 24. The total cpu time used by the method was 13.84 seconds, resulting in an average of 0.46 seconds per time step. This method ranked second in terms of cpu time among the direct techniques. Since this method is a direct solution method the cpu time for individual time steps will also be constant. The cpu time per step and the cumulative cpu time used are plotted in Figure 57. This method will always converge to the right solution after the completion of a certain number of calculations. The memory space requirement was 146172 bytes, Table 37.

## 6.5 PSOR Method

This is an iterative method, hence the solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence

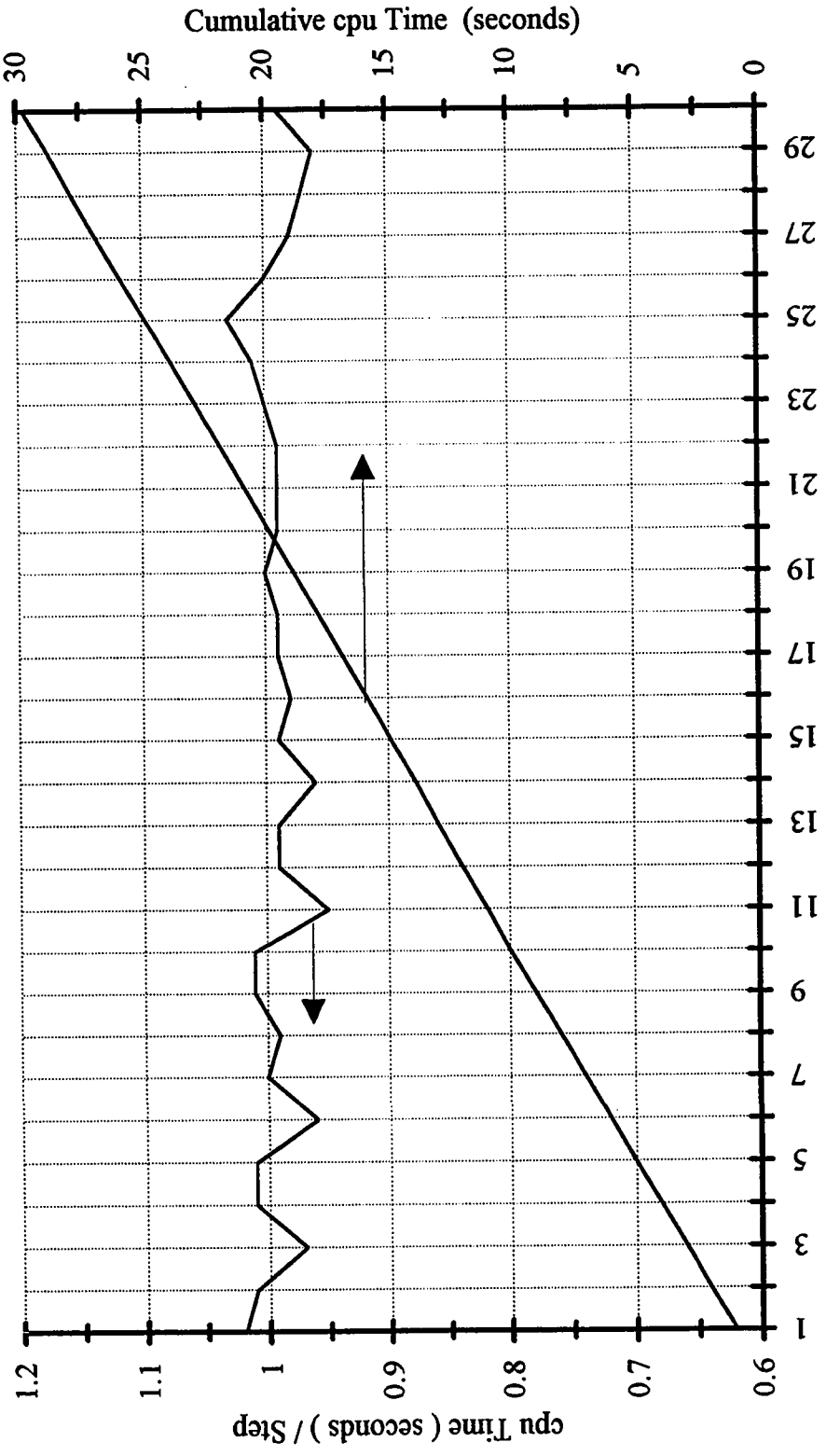



Figure 56 - cpu Time Used by STBAND Method

**TABLE 24 - Simulation Time Steps and cpu Time for RAD Method**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.460 | 0.46 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.480 | 0.94 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.460 | 1.40 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.480 | 1.88 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.470 | 2.35 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.440 | 2.79 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.460 | 3.25 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.460 | 3.71 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.470 | 4.18 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.450 | 4.63 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.440 | 5.07 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.480 | 5.55 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.450 | 6.00 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.440 | 6.44 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.480 | 6.92 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.450 | 7.37 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.470 | 7.84 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.460 | 8.30 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.460 | 8.76 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.460 | 9.22 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.460 | 9.68 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.470 | 10.15 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.480 | 10.63 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.470 | 11.10 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.460 | 11.56 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.450 | 12.01 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.460 | 12.47 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.440 | 12.91 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.470 | 13.38 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.460 | 13.84 |

**AVERAGE CPU TIME / STEP = 13.84 / 30 = 0.46 seconds**

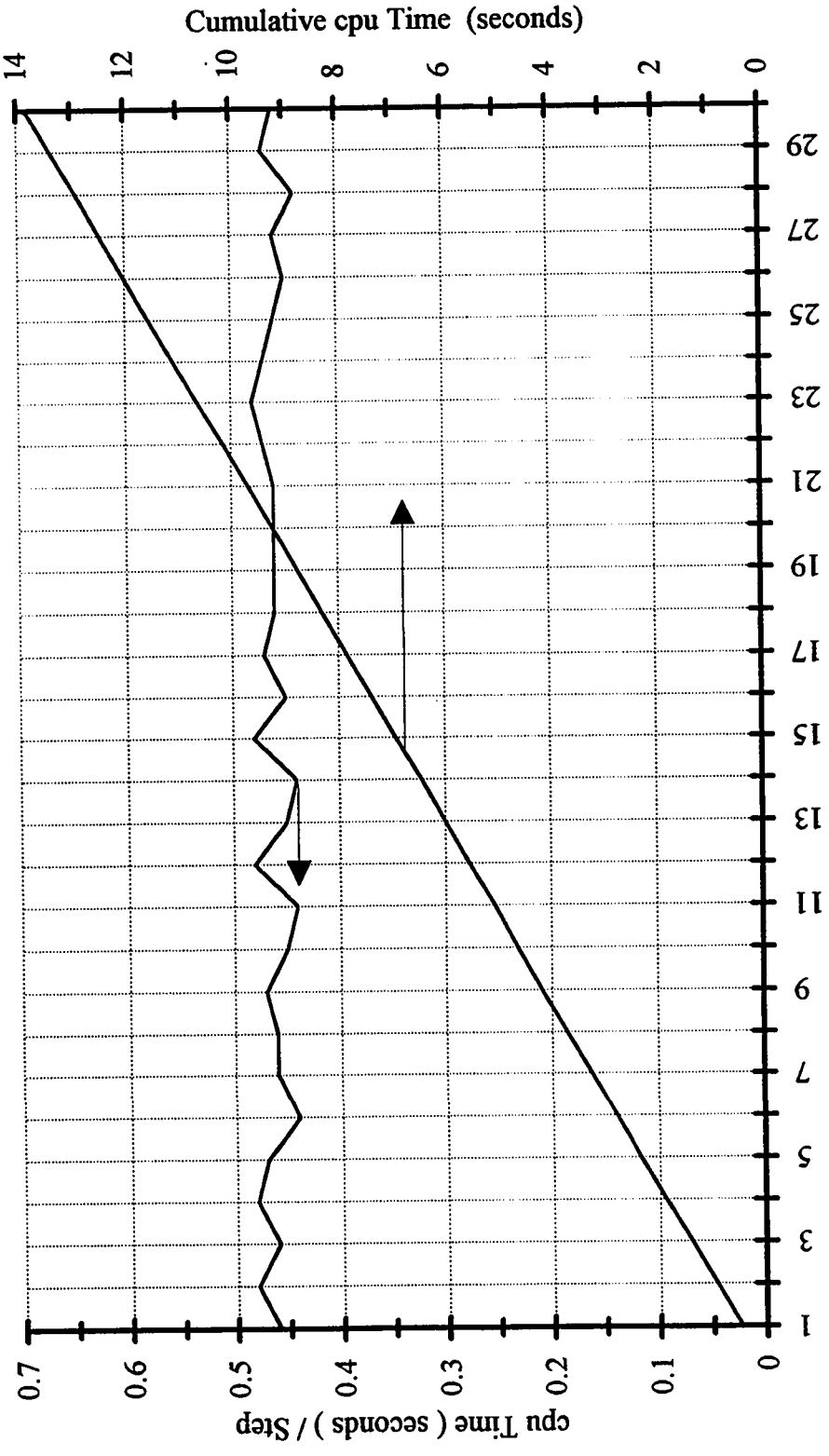



Figure 57 - cpu Time Used by RAD Method

criterion is met. A tolerance of 0.0001 relative change in the pressure was used. This was found to be sufficient to produce accurate pressures within a 0.1 psi difference compared to those obtained with the direct methods.

The difficulty in this method was to find the optimum relaxation parameter which is very detrimental to the performance of the method. Several runs were made in which different parameters between 1 and 2 were tested. The results were plotted and the parameter which produced the least cpu time which was 1.79 was used as the optimum relaxation parameter. A listing of the computer program for this method is attached in the appendix.

The time steps and the relevant cpu time consumption for this method are listed in Table 25. The total cpu time used was 10.89 seconds, resulting in an average of 0.36 seconds per time step. This method was found to be the most efficient in terms of both the cpu time and the memory space requirement among the iterative techniques. Since this method is an iterative solution method the cpu time for individual time steps will vary according to the number of iterations it takes to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 58.

The iterations summary of this method is listed in Table 26, and is plotted in Figure 59. This method performance will vary according to the simulation problem at hand. Since it is an iterative procedure, for some reservoir problems convergence is not guaranteed. Another disadvantage of this method is that its performance is dependent on the relaxation parameter. Hence, any change in the reservoir problem will require a new optimum relaxation parameter to be obtained. The memory space requirement for this method was 7916 bytes, Table 37.

**TABLE 25 - Simulation Time Steps and cpu Time for PSOR Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.390 | 0.39 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.570 | 0.96 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.500 | 1.46 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.450 | 1.91 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.390 | 2.30 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.390 | 2.69 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.390 | 3.08 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.360 | 3.44 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.320 | 3.76 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.310 | 4.07 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.290 | 4.36 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.300 | 4.66 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.360 | 5.02 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.290 | 5.31 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.350 | 5.66 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.380 | 6.04 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.310 | 6.35 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.340 | 6.69 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.380 | 7.07 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.310 | 7.38 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.340 | 7.72 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.420 | 8.14 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.330 | 8.47 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.370 | 8.84 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.380 | 9.22 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.280 | 9.50 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.360 | 9.86 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.380 | 10.24 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.290 | 10.53 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.360 | 10.89 |

**AVERAGE CPU TIME / STEP = 10.89 / 30 0.36 seconds**

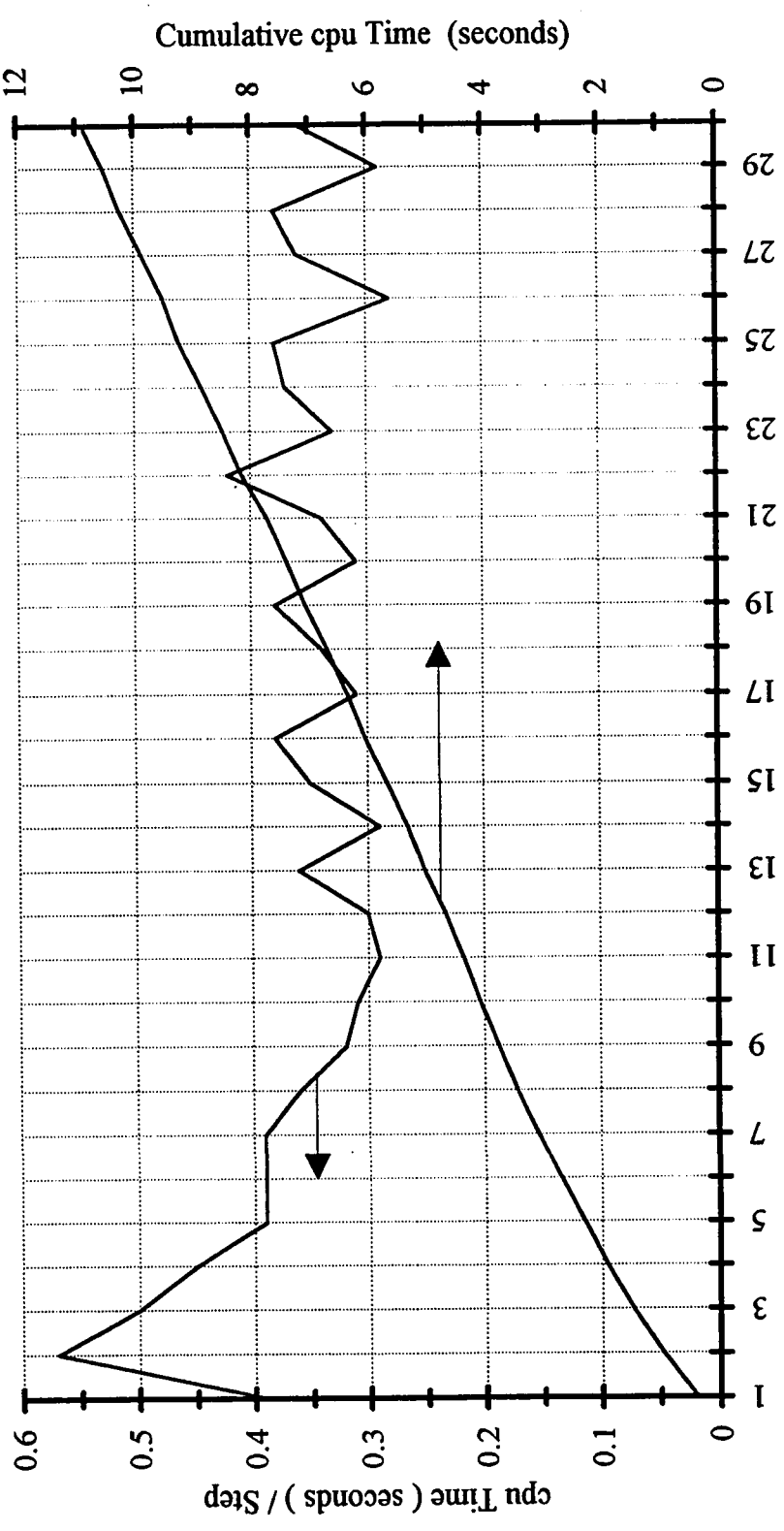



Figure 58 - cpu Time Used by PSOR Method

**TABLE 26 - Iteration Summary for PSOR Method**

| TIME STEP NUMBER | NO. OF ITERATIONS | ITERATION PARAMETER $\omega$ |
|---|---|---|
| 1 | 56 | 1.79 |
| 2 | 86 | 1.79 |
| 3 | 74 | 1.79 |
| 4 | 66 | 1.79 |
| 5 | 57 | 1.79 |
| 6 | 55 | 1.79 |
| 7 | 55 | 1.79 |
| 8 | 50 | 1.79 |
| 9 | 41 | 1.79 |
| 10 | 43 | 1.79 |
| 11 | 38 | 1.79 |
| 12 | 41 | 1.79 |
| 13 | 56 | 1.79 |
| 14 | 38 | 1.79 |
| 15 | 48 | 1.79 |
| 16 | 55 | 1.79 |
| 17 | 40 | 1.79 |
| 18 | 48 | 1.79 |
| 19 | 55 | 1.79 |
| 20 | 40 | 1.79 |
| 21 | 48 | 1.79 |
| 22 | 57 | 1.79 |
| 23 | 43 | 1.79 |
| 24 | 52 | 1.79 |
| 25 | 56 | 1.79 |
| 26 | 39 | 1.79 |
| 27 | 49 | 1.79 |
| 28 | 55 | 1.79 |
| 29 | 39 | 1.79 |
| 30 | 48 | 1.79 |

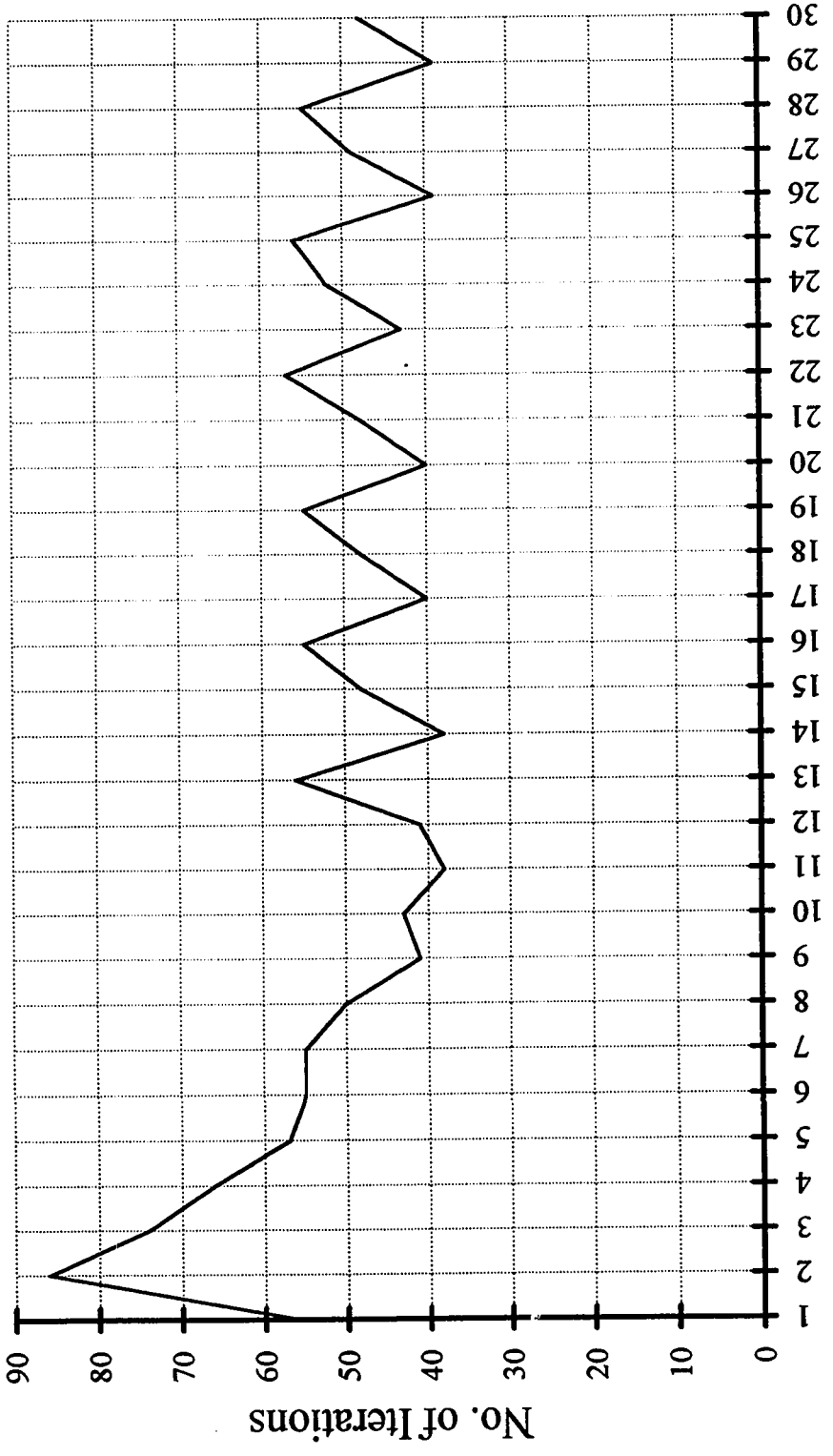



Figure 59 - Iterations vs Time Steps of PSOR Method

## 6.6 LSOR Method

This is an iterative method in which the solution is obtained for individual rows successively untill the whole grid is swept. **Gaussian Elimination** procedure was implemented in solving for the pressures for the small tridiagonal matrices created for the individual rows. It was used instead of the **Thomas Algorithm** since it was found to require less cpu time. A listing of the computer program for this method is attached in the appendix.

Just like **PSOR**, the solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. A tolerance of 0.0001 relative change in the pressure was also used. This was found to be sufficient to produce accurate pressures within a 0.1 psi difference compared to those obtained with the direct methods.

Also, a difficulty in this method was finding the optimum relaxation parameter which is important to the performance of the method. Several runs were made in which $\omega$ was varied between 1 and 2. The results were plotted and the parameter producing the least cpu time which is 1.71 was used as the optimum relaxation parameter.

The time steps and the relevant cpu time consumption for this method are listed in Table 27. The total cpu time used was 11.74 seconds, resulting in an average of 0.39 seconds per time step. Since this method is an iterative solution method the cpu time for individual time steps will vary according to the number of iterations it takes to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 60.

The iterations summary of this method is listed in Table 28, and is plotted in Figure 61. The method took less iterations to converge than **PSOR**. However, since the iteration time for this method was higher than that of **PSOR**, the cpu

**TABLE 27 - Simulation Time Steps and cpu Time for LSOR Method.**

| TIME STEP NUMBER | TIME [ days] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.420 | 0.42 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.620 | 1.04 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.570 | 1.61 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.480 | 2.09 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.430 | 2.52 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.430 | 2.95 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.400 | 3.35 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.390 | 3.74 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.320 | 4.06 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.310 | 4.37 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.320 | 4.69 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.330 | 5.02 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.420 | 5.44 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.320 | 5.76 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.360 | 6.12 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.430 | 6.55 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.340 | 6.89 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.360 | 7.25 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.440 | 7.69 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.340 | 8.03 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.340 | 8.37 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.430 | 8.80 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.350 | 9.15 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.380 | 9.53 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.430 | 9.96 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.340 | 10.30 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.350 | 10.65 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.430 | 11.08 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.320 | 11.40 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.340 | 11.74 |

**AVERAGE CPU TIME / STEP =11.74 / 30 = 0.39 seconds**

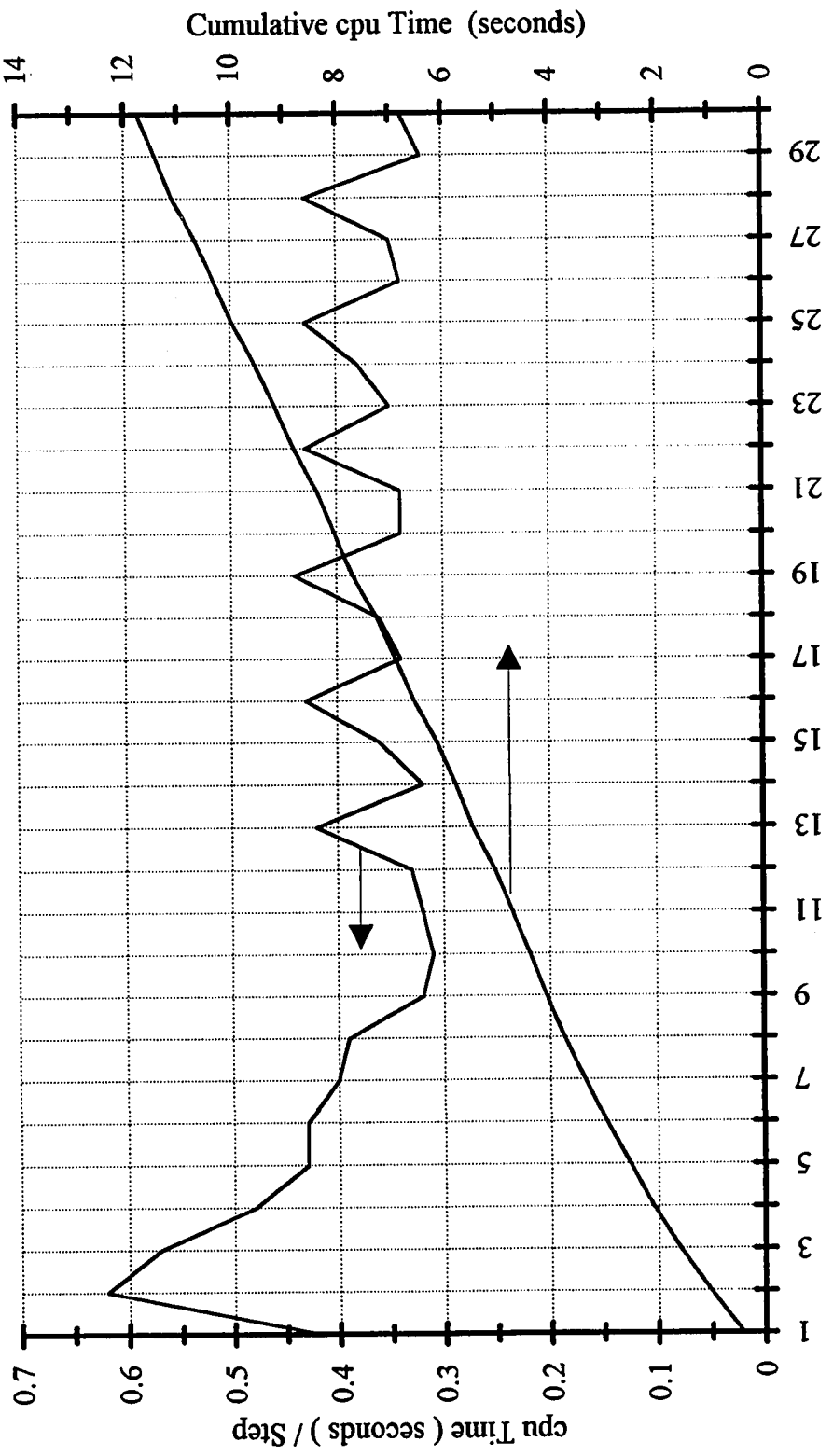



Figure 60 - cpu Time Used by LSOR Method

**TABLE 28 - Iteration Summary for LSOR Method.**

| TIME STEP NUMBER | NO. OF ITERATIONS | ITERATION PARAMETER $\omega$ |
|---|---|---|
| 1 | 40 | 1.71 |
| 2 | 63 | 1.71 |
| 3 | 57 | 1.71 |
| 4 | 49 | 1.71 |
| 5 | 42 | 1.71 |
| 6 | 42 | 1.71 |
| 7 | 38 | 1.71 |
| 8 | 37 | 1.71 |
| 9 | 31 | 1.71 |
| 10 | 29 | 1.71 |
| 11 | 30 | 1.71 |
| 12 | 31 | 1.71 |
| 13 | 41 | 1.71 |
| 14 | 28 | 1.71 |
| 15 | 32 | 1.71 |
| 16 | 42 | 1.71 |
| 17 | 30 | 1.71 |
| 18 | 32 | 1.71 |
| 19 | 42 | 1.71 |
| 20 | 30 | 1.71 |
| 21 | 32 | 1.71 |
| 22 | 42 | 1.71 |
| 23 | 32 | 1.71 |
| 24 | 36 | 1.71 |
| 25 | 42 | 1.71 |
| 26 | 30 | 1.71 |
| 27 | 32 | 1.71 |
| 28 | 41 | 1.71 |
| 29 | 30 | 1.71 |
| 30 | 32 | 1.71 |

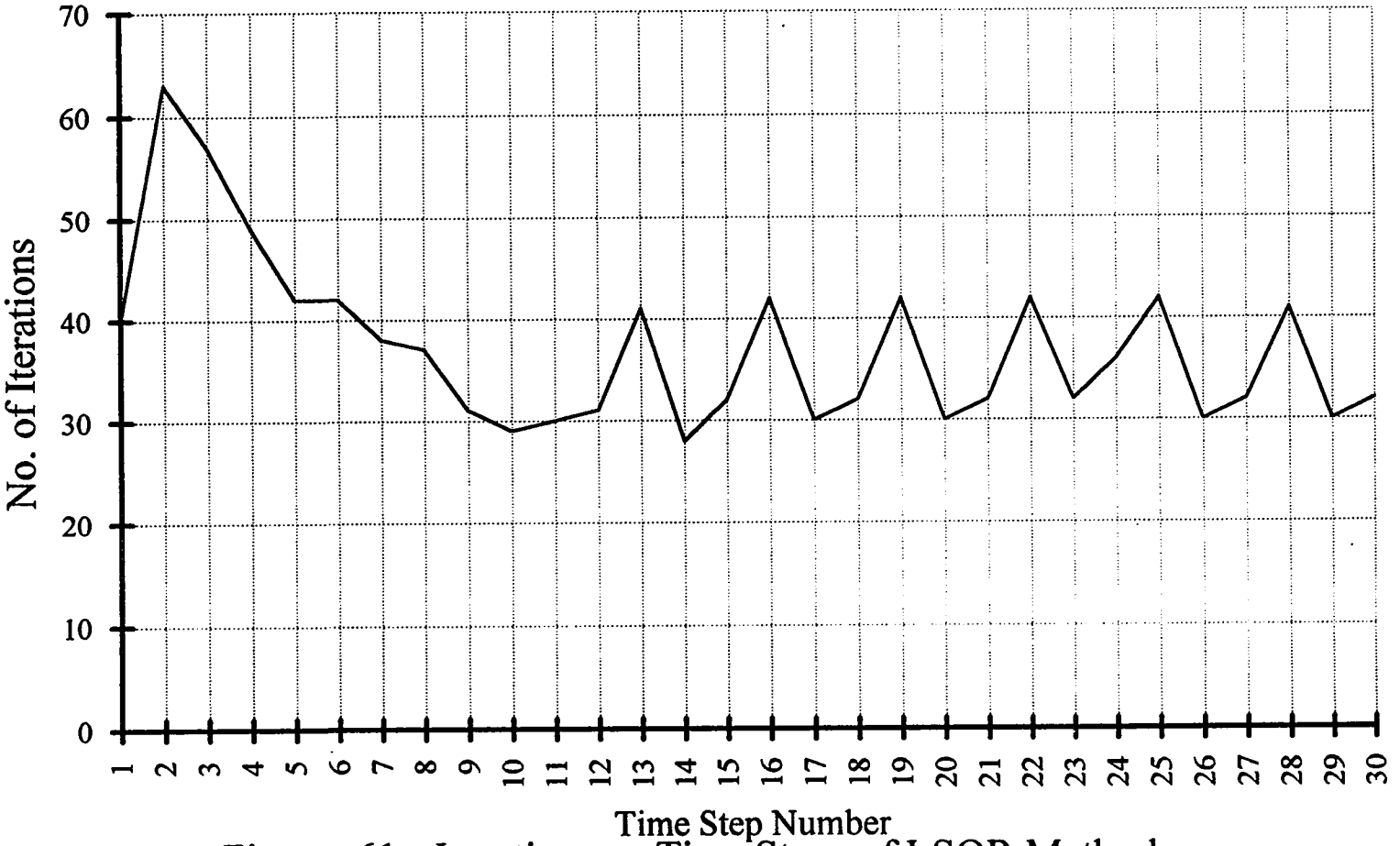



Figure 61 - Iterations vs Time Steps of LSOR Method

time per step became higher.

The performance of this method is also expected to vary for other simulation problems. Since it is an iterative procedure in some reservoir problems it may not converge. However, this method may converge in certain problems in which **PSOR** method may fail to converge.

Just similar to **PSOR**, **LSOR** performance is dependent on the relaxation parameter. Any change in the reservoir problem will also require a new optimum relaxation parameter to be calculated. The memory space requirement was 14536 bytes, Table 37.

## 6.7 ADIP Method

This is an iterative method in which the solution is obtained for individual rows in a horizontal sweep and then for every column in a vertical sweep, successively. **Gaussian Elimination** procedure was implemented in solving for the pressures for the small tridiagonal matrices created for the individual rows or columns. Again, it was used in this method instead of the **Thomas Algorithm** since it was faster. A listing of the computer program for this method is attached in the appendix.

Just like **LSOR**, the solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. A tolerance of 0.0001 relative change in the pressure was used. Again, the difficulty in this method was finding the series of optimum relaxation parameters which will produce the least cpu time. This method was tested with a single parameter only. Similar to **PSOR** and **LSOR**, several runs were made in which $\omega$ was varied between 1 and 2. The results were plotted and the parameter which produced the least cpu time which was 1.77 was used as the optimum relaxation

parameter.

The time steps and the relevant cpu time consumption for this method are listed in Table 29. The total cpu time used by the method was 14.98 seconds, resulting in an average of 0.50 seconds per time step. Since this method is an iterative solution method the cpu time for individual time steps will vary according to the number of iterations it takes to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 62.

The iterations summary of this method is listed in Table 30, and is plotted in Figure 63. The method took less iterations to converge than **LSOR** because it iterates on the problem in two directions, once every row and then every column. This will cause somehow a faster convergence. However, since the cpu time per step per iteration for this method was higher than that of **LSOR**, the total cpu time consumed became higher than that of **LSOR**.

**ADIP** performance will vary according to the simulation problem at hand. Since it is an iterative procedure, in certain circumstances it may not converge to the right solution. However, it could converge in certain problems in which the **SOR** methods do not converge.

One of its disadvantages is that its performance is dependent on a series of relaxation parameters which are difficult to find. And with any change in the reservoir problem estimation of new optimum relaxation parameters will be required. The memory space requirement for this method was 15776 bytes, Table 37.

## 6.8 SIP Method

This is an iterative method in which the original matrix is modified and then easily factored into two submatrices, a lower and an upper triangular

**TABLE 29 - Simulation Time Steps and cpu Time for ADIP Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.260 | 0.26 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.450 | 0.71 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.490 | 1.20 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.580 | 1.78 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.640 | 2.42 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.720 | 3.14 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.640 | 3.78 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.570 | 4.35 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.440 | 4.79 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.420 | 5.21 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.440 | 5.65 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.460 | 6.11 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.560 | 6.67 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.410 | 7.08 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.460 | 7.54 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.590 | 8.13 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.450 | 8.58 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.470 | 9.05 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.560 | 9.61 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.440 | 10.05 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.470 | 10.52 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.550 | 11.07 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.460 | 11.53 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.490 | 12.02 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.550 | 12.57 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.440 | 13.01 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.470 | 13.48 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.570 | 14.05 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.460 | 14.51 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.470 | 14.98 |

**AVERAGE CPU TIME / STEP = 14.98 / 30 = 0.50 seconds**

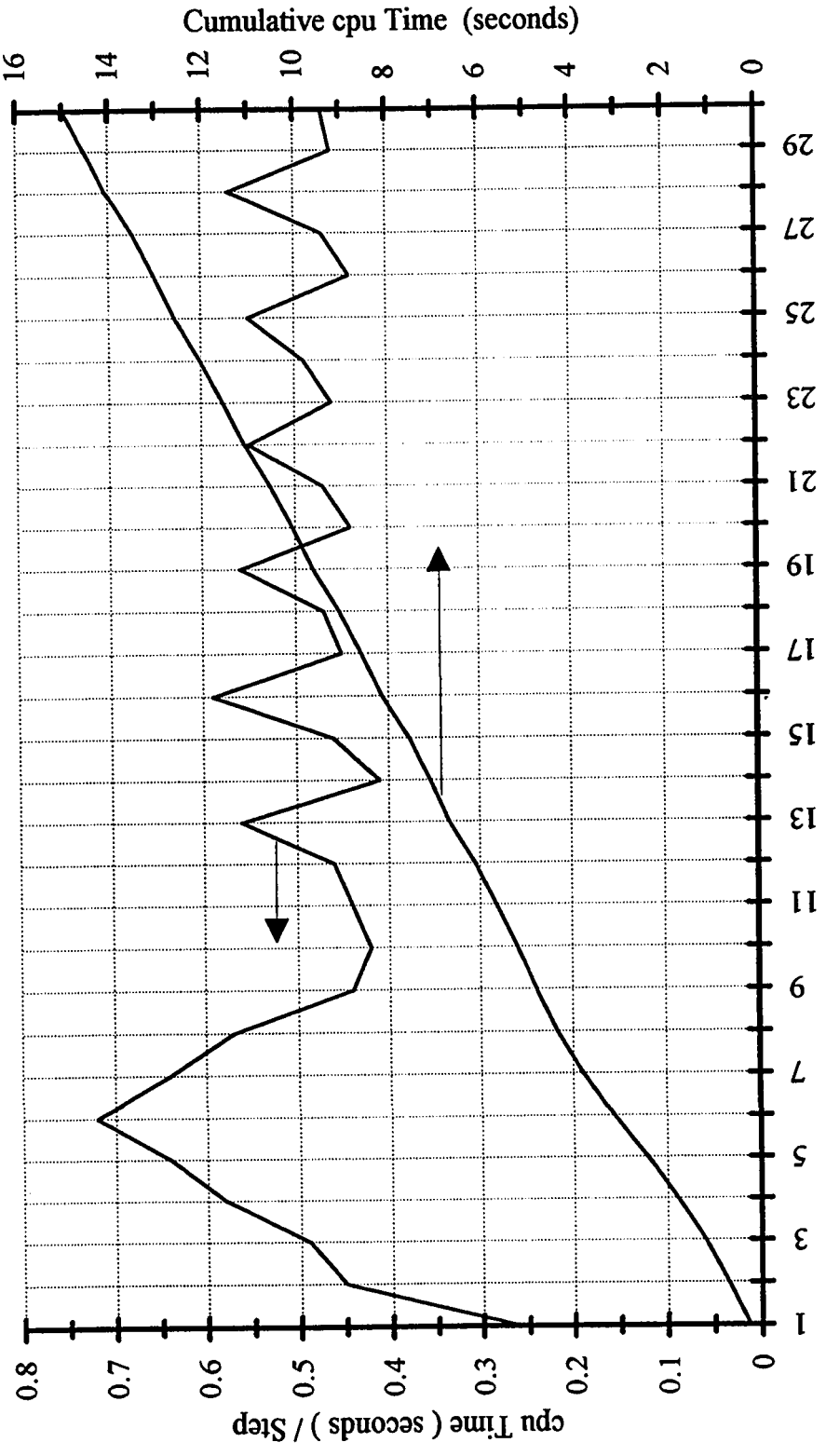



Figure 62 - cpu Time Used by ADIP Method

**TABLE 30 - Iteration Summary for ADIP Method.**

| TIME STEP NUMBER | NO. OF ITERATIONS | ITERATION PARAMETER $\omega$ |
|---|---|---|
| 1 | 12 | 1.77 |
| 2 | 26 | 1.77 |
| 3 | 27 | 1.77 |
| 4 | 32 | 1.77 |
| 5 | 38 | 1.77 |
| 6 | 41 | 1.77 |
| 7 | 38 | 1.77 |
| 8 | 33 | 1.77 |
| 9 | 26 | 1.77 |
| 10 | 23 | 1.77 |
| 11 | 25 | 1.77 |
| 12 | 26 | 1.77 |
| 13 | 32 | 1.77 |
| 14 | 23 | 1.77 |
| 15 | 26 | 1.77 |
| 16 | 32 | 1.77 |
| 17 | 24 | 1.77 |
| 18 | 26 | 1.77 |
| 19 | 32 | 1.77 |
| 20 | 24 | 1.77 |
| 21 | 26 | 1.77 |
| 22 | 32 | 1.77 |
| 23 | 25 | 1.77 |
| 24 | 27 | 1.77 |
| 25 | 32 | 1.77 |
| 26 | 24 | 1.77 |
| 27 | 26 | 1.77 |
| 28 | 32 | 1.77 |
| 29 | 24 | 1.77 |
| 30 | 26 | 1.77 |

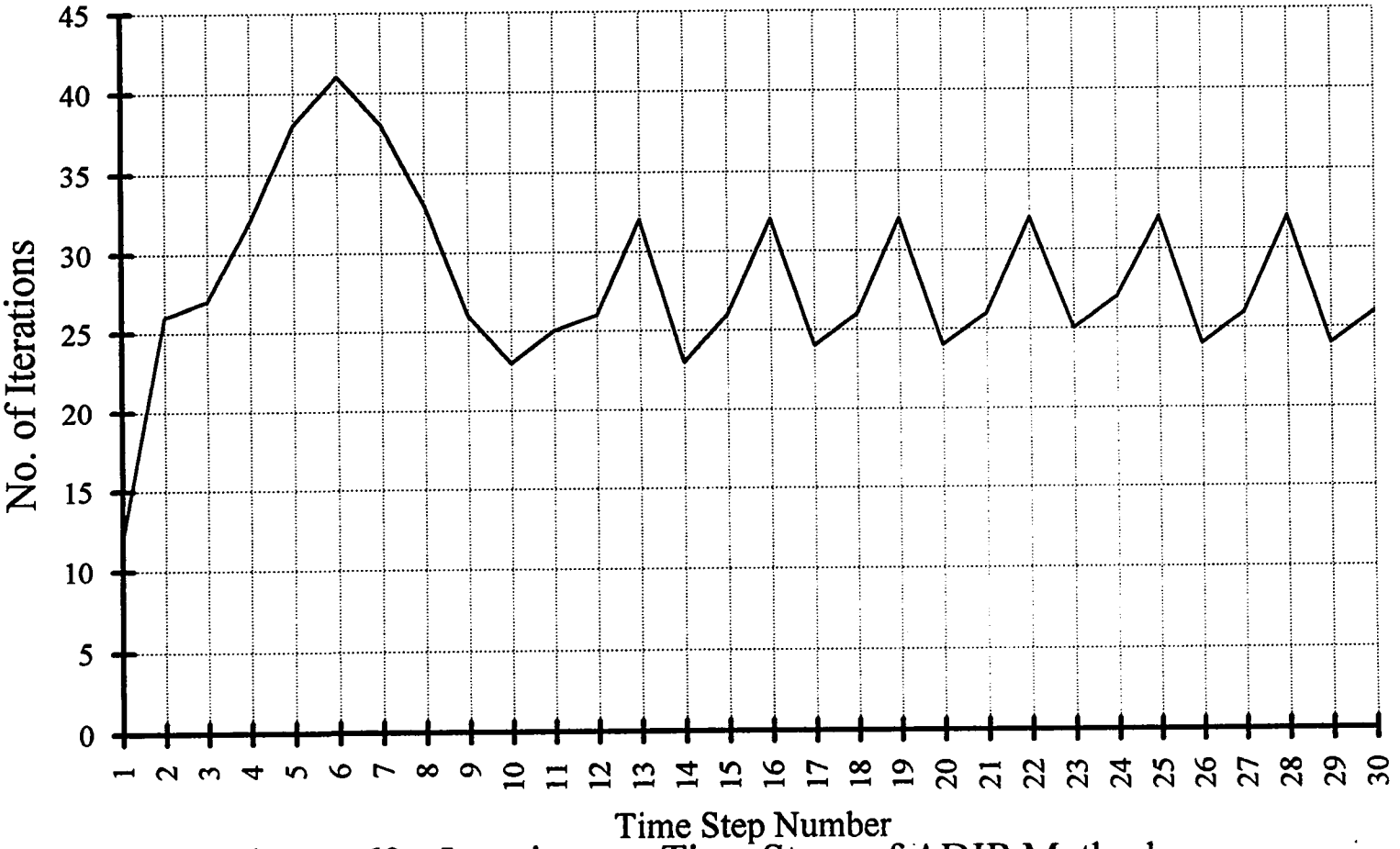



Figure 63 - Iterations vs Time Steps of ADIP Method

matrices. A listing of the computer program for this method is attached in the appendix.

Like **ADIP**, the solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. A tolerance of 0.0001 relative change in the pressure was used. Again, the difficulty in this method was finding the series of optimum iteration parameters which will produce the least time. This method was tested with a single parameter only. Similar to **PSOR** and **LSOR**, several runs were made in which $\alpha$ was varied between 0 and 2. The results were plotted and the parameter which produced the least cpu time was 0.985 and was chosen as the optimum iteration parameter.

The time steps and the relevant cpu time consumption for this method are listed in Table 31. The total cpu time used was 20.53 seconds, resulting in an average of 0.68 seconds per time step. Since this method is an iterative solution method the cpu time for individual time steps will vary according to the number of iterations it takes to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 64.

Its iteration summary is listed in Table 32, and plotted in Figure 65. The method took more iterations to converge than **ADIP.**

This method performance will vary according to the simulation problem at hand. Since it is an iterative procedure in some circumstances it may not converge to the right solution. However, this method may converge in certain problems in which **SOR** or **ADIP** methods do not converge.

Again a disadvantage of this method is that its performance is dependent on a series of relaxation parameters. Thus any change in the reservoir problem will require new optimum relaxation parameters to be calculated. The memory space requirement was 30492 bytes, Table 37.

**TABLE 31 - Simulation Time Steps and cpu Time for SIP Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw (fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.330 | 0.33 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.600 | 0.93 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.610 | 1.54 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.660 | 2.20 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.940 | 3.14 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.910 | 4.05 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.770 | 4.82 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.640 | 5.46 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.690 | 6.15 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.690 | 6.84 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.620 | 7.46 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.750 | 8.21 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.670 | 8.88 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.610 | 9.49 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.700 | 10.19 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.730 | 10.92 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.630 | 11.55 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.700 | 12.25 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.670 | 12.92 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.620 | 13.54 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.690 | 14.23 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.760 | 14.99 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.620 | 15.61 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.720 | 16.33 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.780 | 17.11 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.650 | 17.76 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.710 | 18.47 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.730 | 19.20 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.640 | 19.84 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.690 | 20.53 |

**AVERAGE CPU TIME / STEP = 20.53 / 30 = 0.68 seconds**

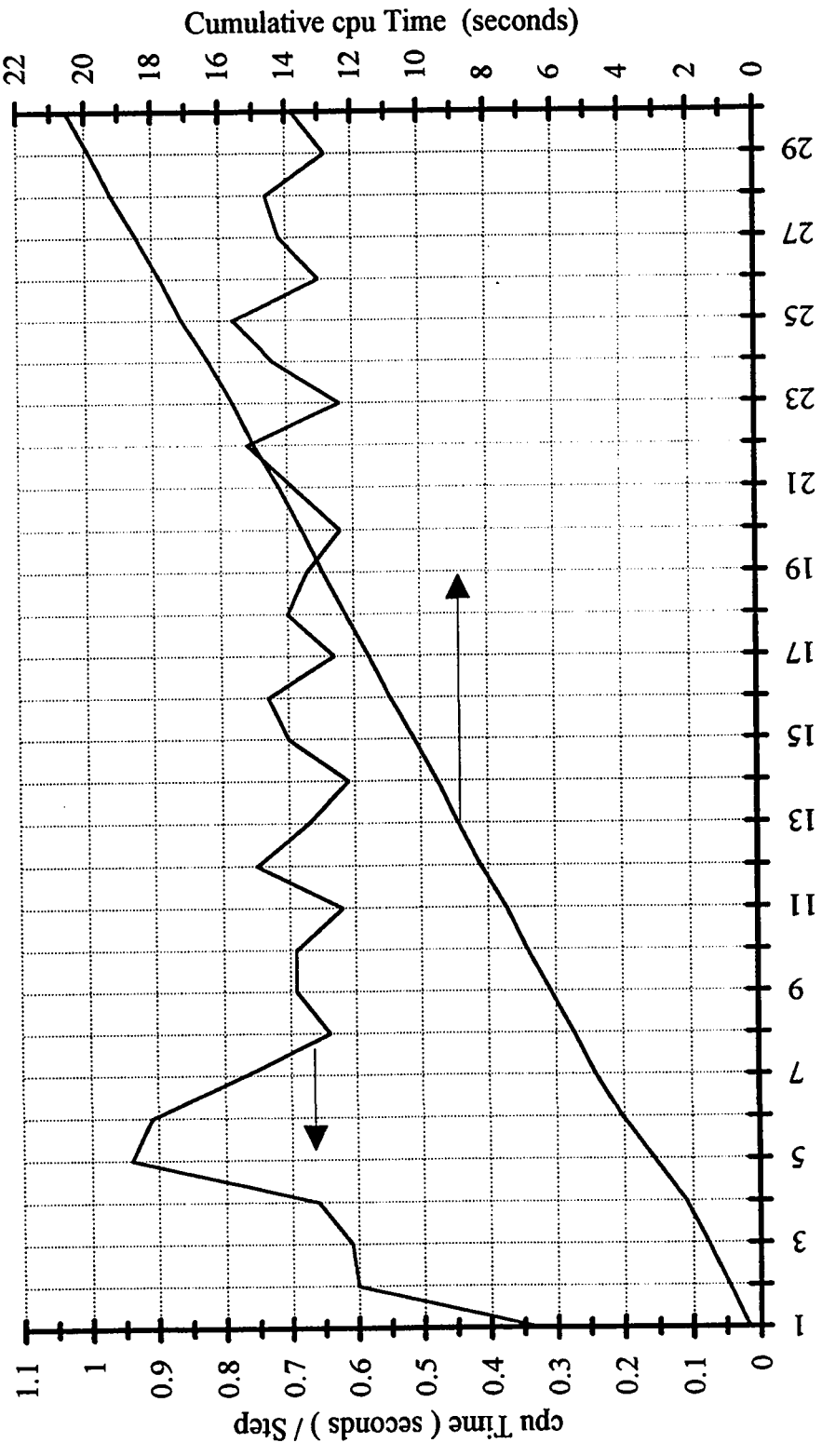



Figure 64 - cpu Time Used by SIP Method

**TABLE 32 - Iteration Summary for SIP Method.**

| TIME STEP NUMBER | NO. OF ITERATIONS | ITERATION PARAMETER $\omega$ |
|---|---|---|
| 1 | 17 | 0.985 |
| 2 | 34 | 0.985 |
| 3 | 35 | 0.985 |
| 4 | 38 | 0.985 |
| 5 | 54 | 0.985 |
| 6 | 55 | 0.985 |
| 7 | 45 | 0.985 |
| 8 | 36 | 0.985 |
| 9 | 39 | 0.985 |
| 10 | 38 | 0.985 |
| 11 | 34 | 0.985 |
| 12 | 42 | 0.985 |
| 13 | 39 | 0.985 |
| 14 | 34 | 0.985 |
| 15 | 41 | 0.985 |
| 16 | 44 | 0.985 |
| 17 | 35 | 0.985 |
| 18 | 40 | 0.985 |
| 19 | 38 | 0.985 |
| 20 | 35 | 0.985 |
| 21 | 40 | 0.985 |
| 22 | 46 | 0.985 |
| 23 | 36 | 0.985 |
| 24 | 43 | 0.985 |
| 25 | 45 | 0.985 |
| 26 | 36 | 0.985 |
| 27 | 41 | 0.985 |
| 28 | 43 | 0.985 |
| 29 | 36 | 0.985 |
| 30 | 40 | 0.985 |

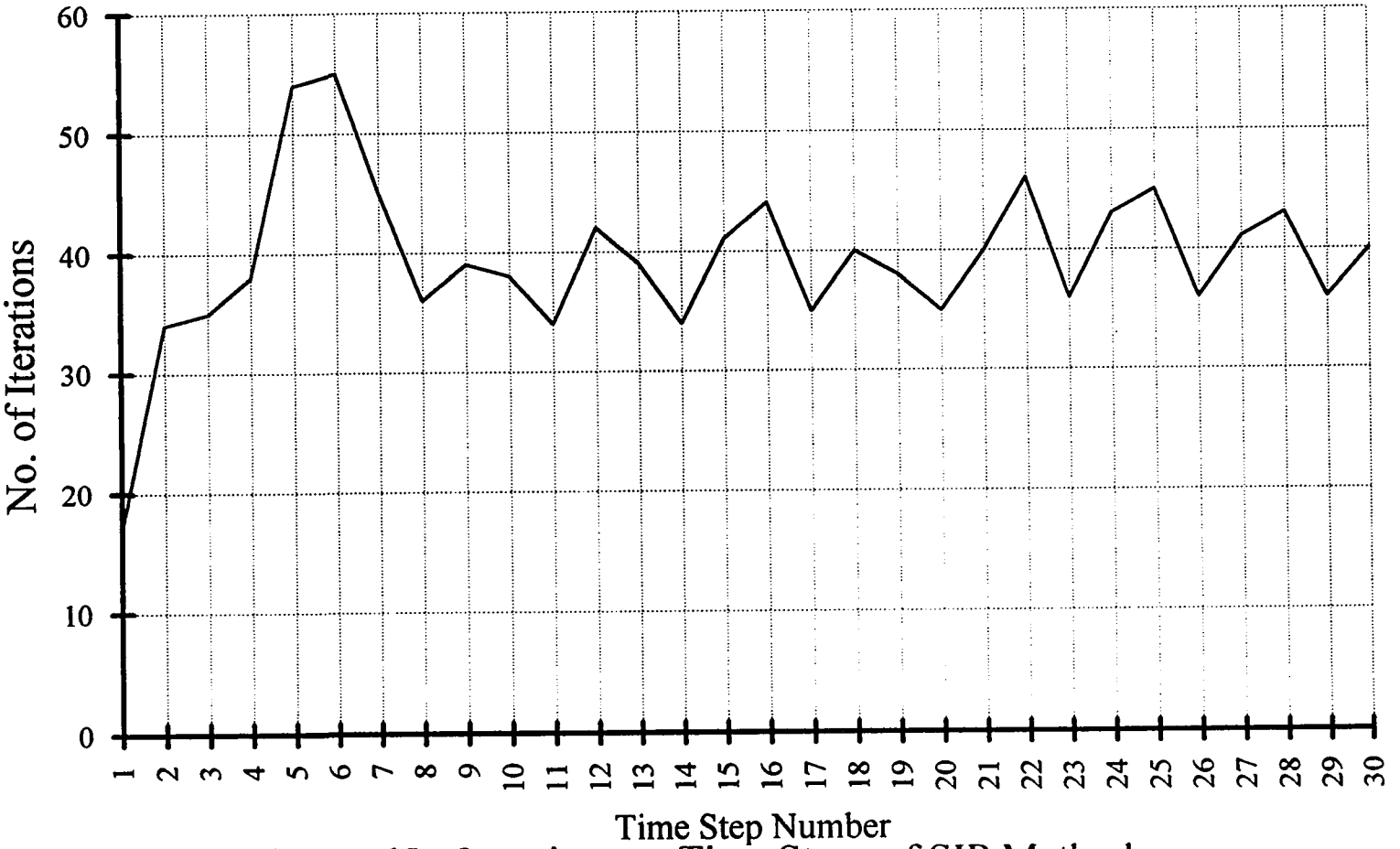



Figure 65 - Iterations vs Time Steps of SIP Method

## 6.9 CGTD Method

This is an iterative procedure in which a preconditioning matrix is formed for use with the conjugate gradient method. This method does not require any iteration parameters which makes it superior in this respect to the other iterative methods discussed before. A listing of the computer program for this method is attached in the appendix.

The solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. A relative error of 0.0001 in the pressures was used.

The time steps and the relevant cpu time consumption for this method are listed in Table 33. The total cpu time used by the method was 23.37 seconds, resulting in an average of 0.78 seconds per time step. This method is an iterative solution method and therefore the cpu time for individual time steps varied according to the number of iterations it took to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 66.

Its iteration summary is listed in Table 34, and plotted in Figure 67. This method primarily makes use of the conjugate gradient method and its performane will be mainly dependent on how good is the approximate inverse of the approximate symmetric matrix.

An advantage of this method is that its performance is independent of any iteration parameters. Moreover, just like direct solution methods, the convergence of this method is guaranteed. Therefore, any change in the reservoir problem may not affect the performance of this method. The memory space requirement for this method was 112648 bytes, Table 37.

**TABLE 33 - Simulation Time Steps and cpu Time for CGTD Method.**

| TIME STEP NUMBER | TIME [ days] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.510 | 0.51 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.870 | 1.38 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.990 | 2.37 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.710 | 3.08 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.650 | 3.73 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.690 | 4.42 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.710 | 5.13 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.640 | 5.77 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.650 | 6.42 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.680 | 7.10 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.800 | 7.90 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.830 | 8.73 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.890 | 9.62 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.860 | 10.48 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.800 | 11.28 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.820 | 12.10 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.830 | 12.93 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.610 | 13.54 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.870 | 14.41 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.820 | 15.23 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.740 | 15.97 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.930 | 16.90 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.940 | 17.84 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.840 | 18.68 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.840 | 19.52 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.830 | 20.35 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.690 | 21.04 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.780 | 21.82 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.840 | 22.66 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.710 | 23.37 |

**AVERAGE CPU TIME / STEP = 23.37 / 30 = 0.78 seconds**

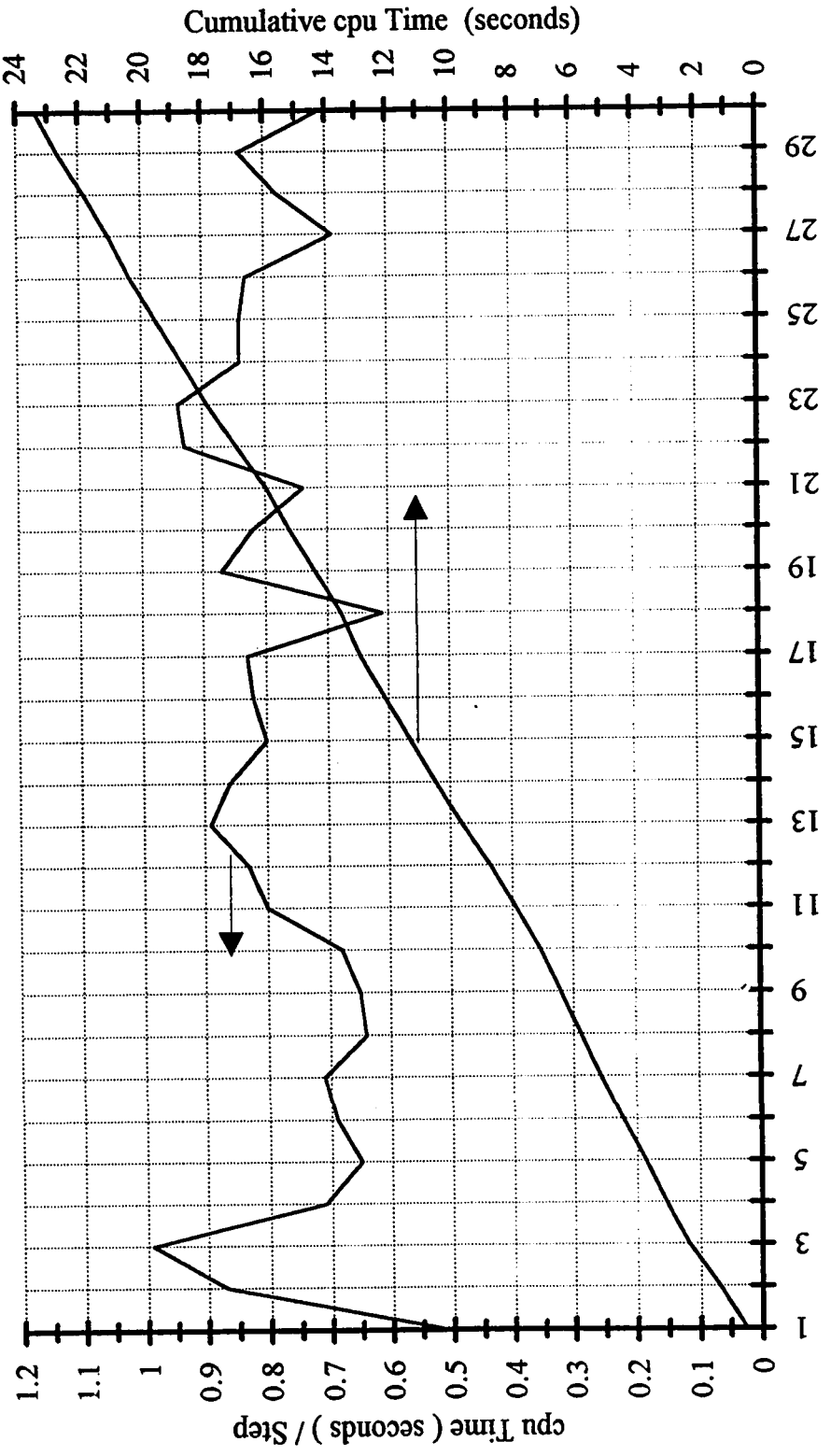



Figure 66 - cpu Time Used by CGTD Method

**TABLE 34 - Iteration Summary for CGTD Method.**

| TIME STEP NUMBER | NO. OF ITERATIONS |
|---|---|
| 1 | 28 |
| 2 | 58 |
| 3 | 64 |
| 4 | 46 |
| 5 | 39 |
| 6 | 44 |
| 7 | 46 |
| 8 | 41 |
| 9 | 41 |
| 10 | 43 |
| 11 | 52 |
| 12 | 54 |
| 13 | 57 |
| 14 | 56 |
| 15 | 52 |
| 16 | 51 |
| 17 | 56 |
| 18 | 37 |
| 19 | 56 |
| 20 | 54 |
| 21 | 46 |
| 22 | 61 |
| 23 | 61 |
| 24 | 55 |
| 25 | 55 |
| 26 | 53 |
| 27 | 44 |
| 28 | 50 |
| 29 | 54 |
| 30 | 44 |

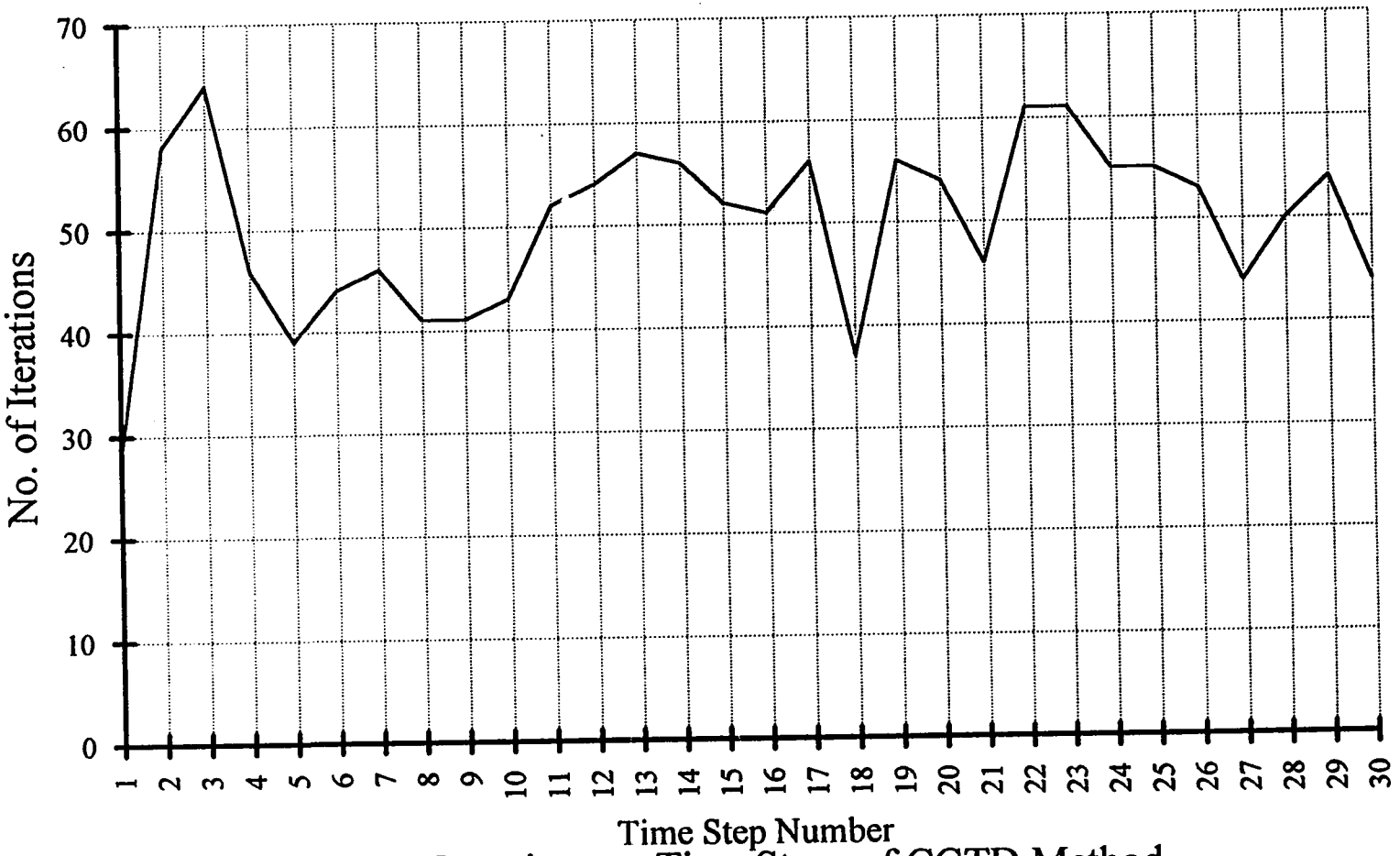



Figure 67 - Iterations vs Time Steps of CGTD Method

## 6.10 NF Method

This is an iterative procedure in which a preconditioning matrix is formed for the conjugate gradient method. Like **CGTD** method, this method does not require any iteration parameters which also makes it superior in this respect to the other iterative methods explained previously. A listing of the computer program for this method is attached in the appendix.

The solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. A relative error in the calculated pressures of 0.0001 was used.

The time steps and the relevant cpu time consumption for this method are listed in Table 35. The total cpu time used by the method was 12.64 seconds, resulting in an average of 0.42 seconds per time step. Since this method is an iterative solution method the cpu time for individual time steps will vary according to the number of iterations it takes to converge. The cpu time per step and the cumulative cpu time used are plotted in Figure 68.

Its iteration summary is listed in Table 36, and is plotted in Figure 69. In comparison to the other iterative methods, this method took the least iterations to converge.

An advantage of this method is that its performance is independent of any iteration parameters. Therefore, any change in the reservoir problem may not affect its performance. The memory space requirement for this method was 26016 bytes as listed in Table 37.

The memory space and the cpu time requirements of all the methods are plotted in Figures 70 and 71, respectively.

It should be noted that the results obtained are pertinent to the test

**TABLE 35 - Simulation Time Steps and cpu Time for NF Method.**

| TIME STEP NUMBER | TIME [ days ] | TIME STEP SIZE [ days ] | MAXIMUM CHANGE IN Sw ( fraction ) | MAXIMUM CHANGE IN PRESSURE (psi ) | CPU TIME [seconds] | CUMULATIVE CPU TIME [seconds] |
|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.10 | -0.005 | 105.17 | 0.320 | 0.32 |
| 2 | 0.25 | 0.15 | -0.007 | 54.72 | 0.370 | 0.69 |
| 3 | 0.50 | 0.25 | -0.011 | 41.98 | 0.430 | 1.12 |
| 4 | 1.00 | 0.50 | -0.023 | 42.48 | 0.450 | 1.57 |
| 5 | 2.00 | 1.00 | -0.044 | 48.34 | 0.500 | 2.07 |
| 6 | 3.00 | 1.00 | -0.041 | 40.84 | 0.450 | 2.52 |
| 7 | 4.25 | 1.25 | -0.047 | 34.51 | 0.440 | 2.96 |
| 8 | 5.00 | 0.75 | -0.025 | 27.87 | 0.380 | 3.34 |
| 9 | 6.00 | 1.00 | -0.030 | 21.43 | 0.380 | 3.72 |
| 10 | 7.25 | 1.25 | -0.033 | 26.47 | 0.410 | 4.13 |
| 11 | 8.81 | 1.56 | -0.033 | 17.92 | 0.420 | 4.55 |
| 12 | 10.00 | 1.19 | -0.018 | 11.39 | 0.410 | 4.96 |
| 13 | 12.00 | 2.00 | -0.030 | 12.14 | 0.440 | 5.40 |
| 14 | 14.00 | 2.00 | -0.031 | 14.94 | 0.440 | 5.84 |
| 15 | 15.00 | 1.00 | -0.015 | 10.26 | 0.410 | 6.25 |
| 16 | 17.00 | 2.00 | -0.029 | 11.36 | 0.420 | 6.67 |
| 17 | 19.00 | 2.00 | -0.028 | 14.16 | 0.430 | 7.10 |
| 18 | 20.00 | 1.00 | -0.013 | 11.09 | 0.420 | 7.52 |
| 19 | 22.00 | 2.00 | -0.025 | 11.81 | 0.430 | 7.95 |
| 20 | 24.00 | 2.00 | -0.022 | 16.41 | 0.460 | 8.41 |
| 21 | 25.00 | 1.00 | -0.010 | 8.71 | 0.400 | 8.81 |
| 22 | 27.00 | 2.00 | -0.018 | 8.02 | 0.420 | 9.23 |
| 23 | 29.00 | 2.00 | -0.019 | 8.92 | 0.410 | 9.64 |
| 24 | 30.00 | 1.00 | -0.009 | 5.85 | 0.420 | 10.06 |
| 25 | 32.00 | 2.00 | -0.019 | 7.18 | 0.410 | 10.47 |
| 26 | 34.00 | 2.00 | -0.019 | 8.61 | 0.450 | 10.92 |
| 27 | 35.00 | 1.00 | -0.009 | 5.78 | 0.390 | 11.31 |
| 28 | 37.00 | 2.00 | -0.018 | 6.93 | 0.450 | 11.76 |
| 29 | 39.00 | 2.00 | -0.018 | 8.45 | 0.450 | 12.21 |
| 30 | 40.00 | 1.00 | -0.008 | 6.17 | 0.430 | 12.64 |

**AVERAGE CPU TIME / STEP = 12.64 / 30 0.42 seconds**

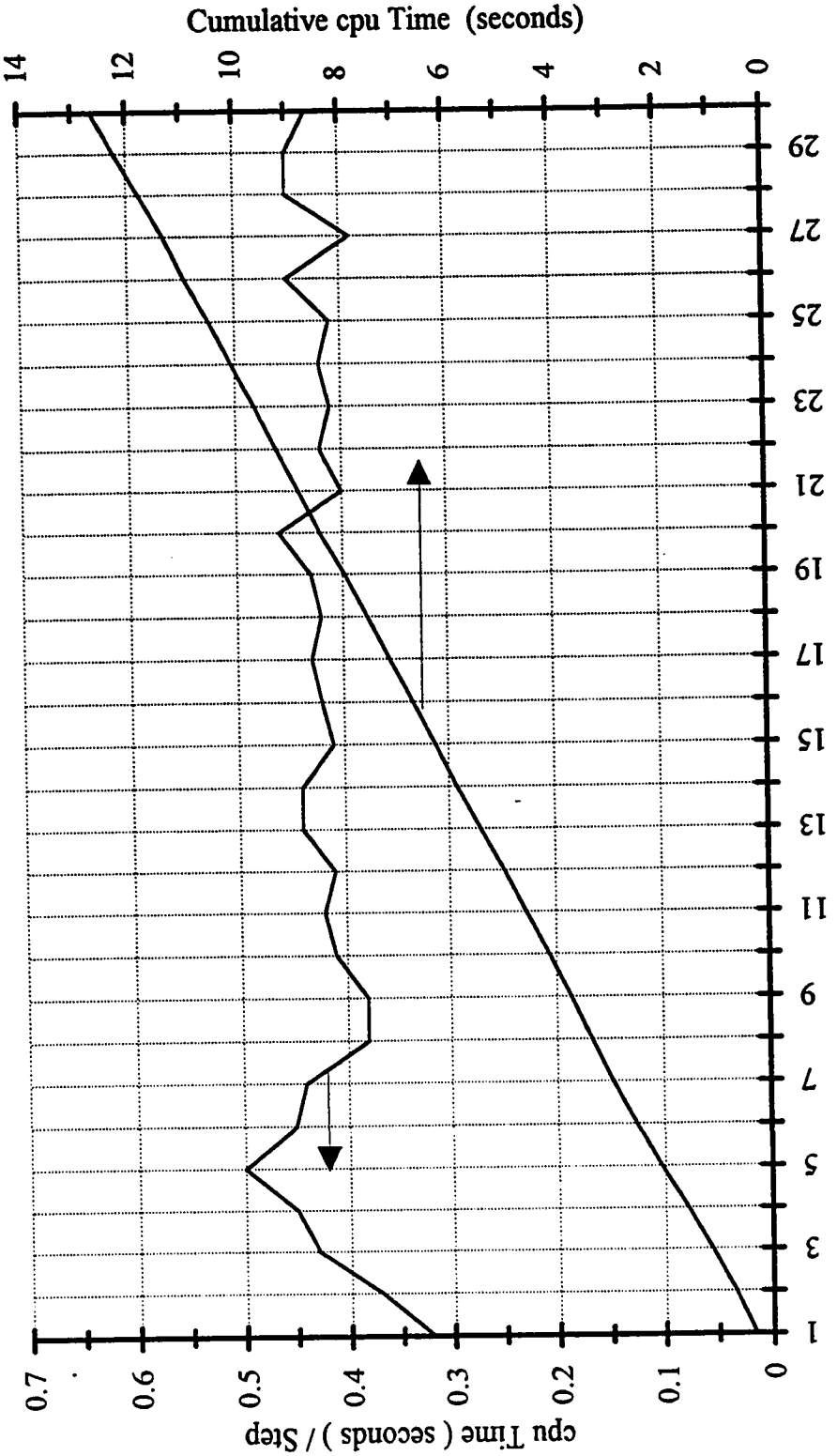



Figure 68 - cpu Time Used by NF Method

**TABLE 36 - Iteration Summary for NF Method.**

| TIME STEP NUMBER | NO. OF ITERATIONS |
|---|---|
| 1 | 9 |
| 2 | 15 |
| 3 | 18 |
| 4 | 18 |
| 5 | 22 |
| 6 | 19 |
| 7 | 20 |
| 8 | 14 |
| 9 | 15 |
| 10 | 16 |
| 11 | 17 |
| 12 | 16 |
| 13 | 18 |
| 14 | 18 |
| 15 | 16 |
| 16 | 18 |
| 17 | 18 |
| 18 | 16 |
| 19 | 18 |
| 20 | 19 |
| 21 | 16 |
| 22 | 18 |
| 23 | 18 |
| 24 | 16 |
| 25 | 17 |
| 26 | 18 |
| 27 | 16 |
| 28 | 18 |
| 29 | 19 |
| 30 | 16 |

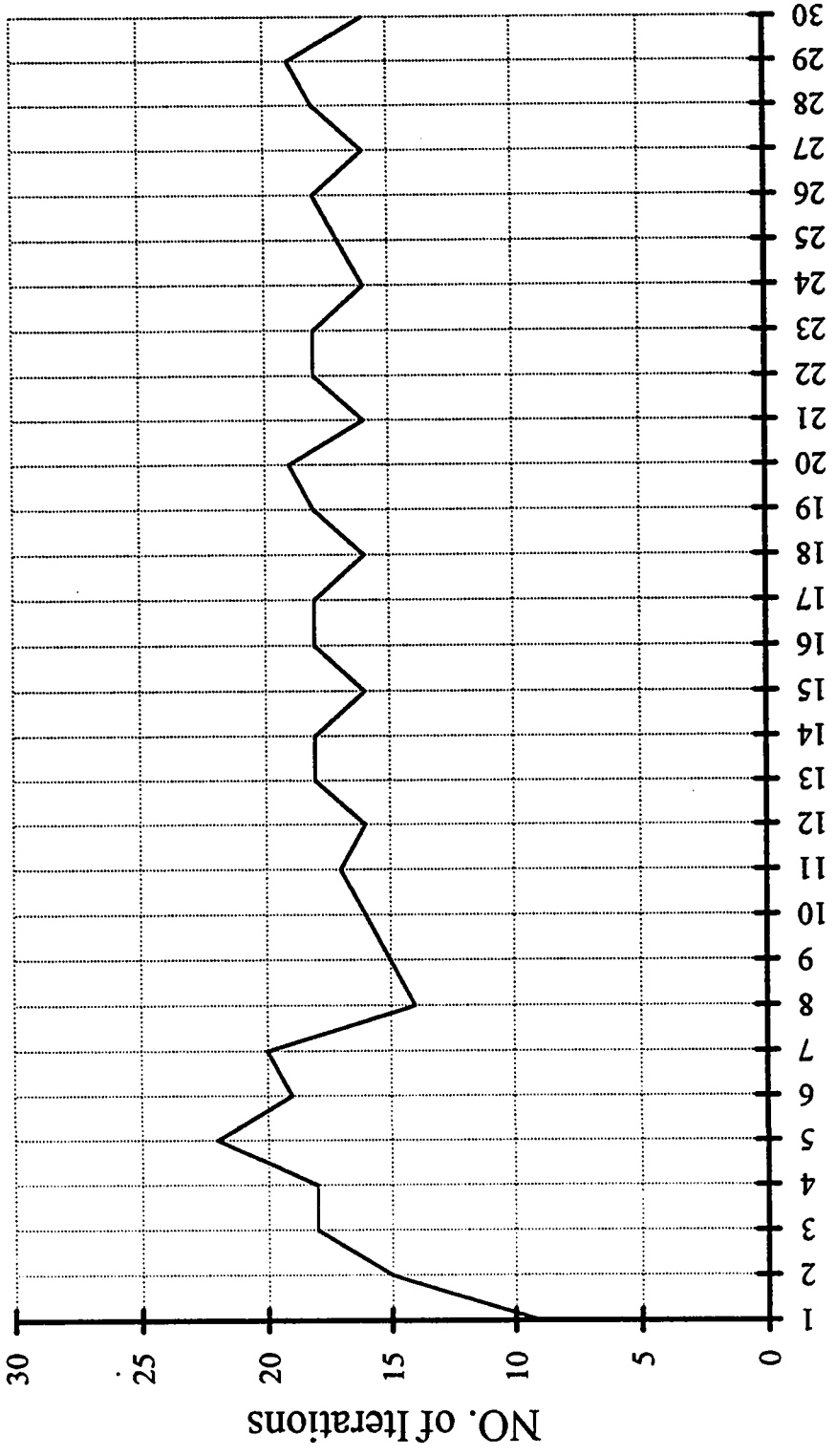



Figure 69 - Iterations vs Time Steps of NF Method

TABLE 37 - Memory Storage and cpu Time for the Methods Tested.

| Method | Memory Storage ( bytes ) | Total cpu Time Used ( seconds ) | Iteration Parameter ω | Average cpu Time per Step ( seconds ) |
|---|---|---|---|---|
| **STANDARD - GAUSS** | 117024 | 19.45 | ---- | 0.65 |
| **D4 - GAUSS** | 105044 | 9.85 | ---- | 0.33 |
| **STBAND** | 180472 | 29.74 | ---- | 0.99 |
| **RAD** | 146172 | 13.84 | ---- | 0.46 |
| **PSOR** | 7916 | 10.89 | 1.79 | 0.36 |
| **LSOR** | 14536 | 11.74 | 1.71 | 0.39 |
| **ADIP** | 15776 | 14.98 | 1.77 | 0.50 |
| **SIP** | 30492 | 20.53 | 0.985 | 0.68 |
| **CGTD** | 112648 | 23.37 | ---- | 0.78 |
| **NF** | 26016 | 12.64 | ----- | 0.42 |

problem chosen for this study which is a 2-D homogeneous five-spot water flooding. Hence, those results and the relevant conclusions from this study should not be generalized for other situations.

The computer cpu time is that of IBM3090. Since these methods were run on the main frame the computation cpu time could have been slightly affected by the time sharing environment rather than on a single user computer. Moreover, the results obtained for the iterative methods are pertinant to a convergence criterion of 0.0001 relative error. Thus, with a different convergence criteria the results will be different.

For other problems which are of different natures, for example, heterogeneous, isotropic, two-phase gas and oil, etc., the results will be different. The problem size will also have an effect. The results will also differ when solving three dimensional problems as well.

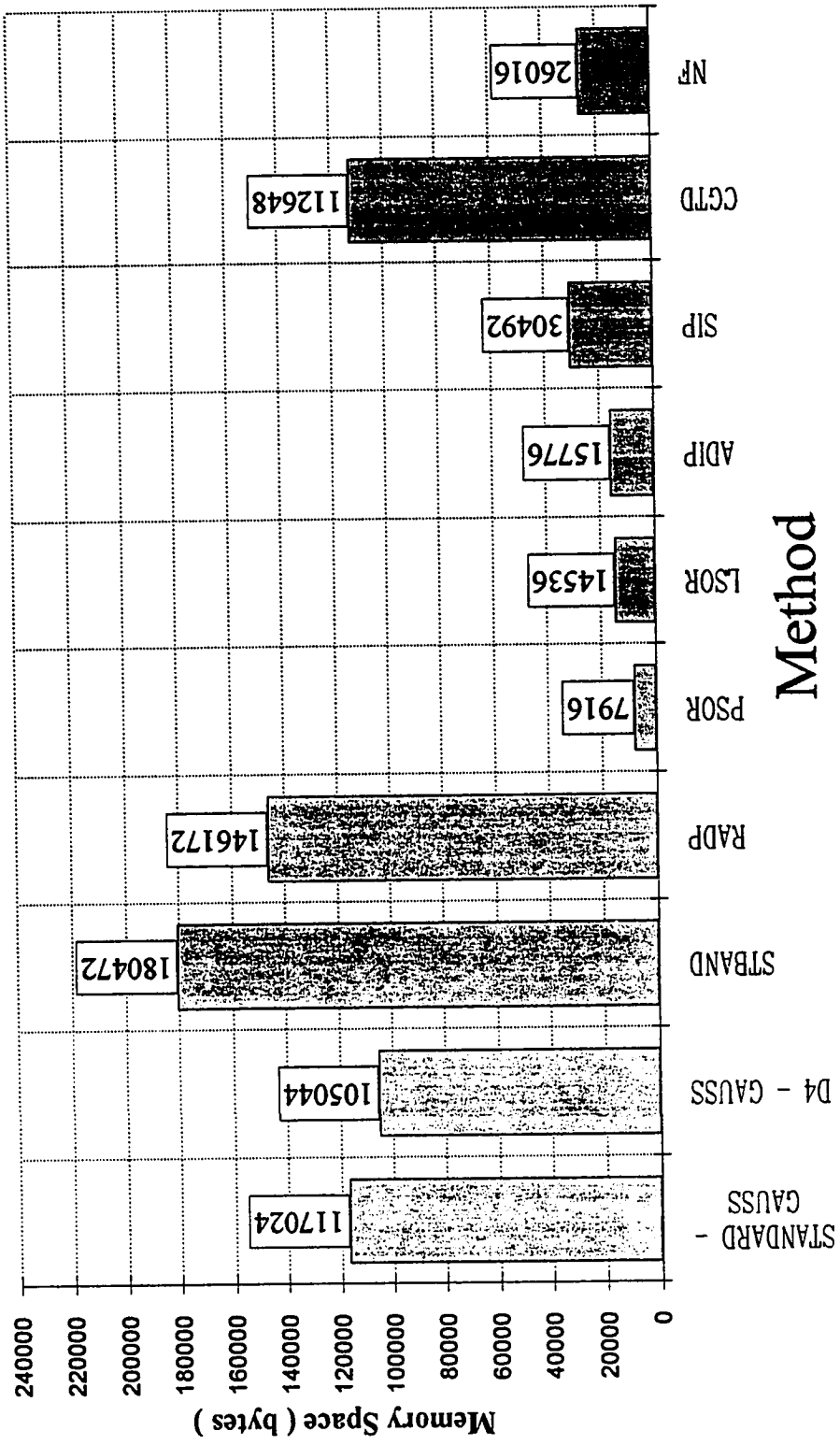



Figure 70 - Storage Requirements

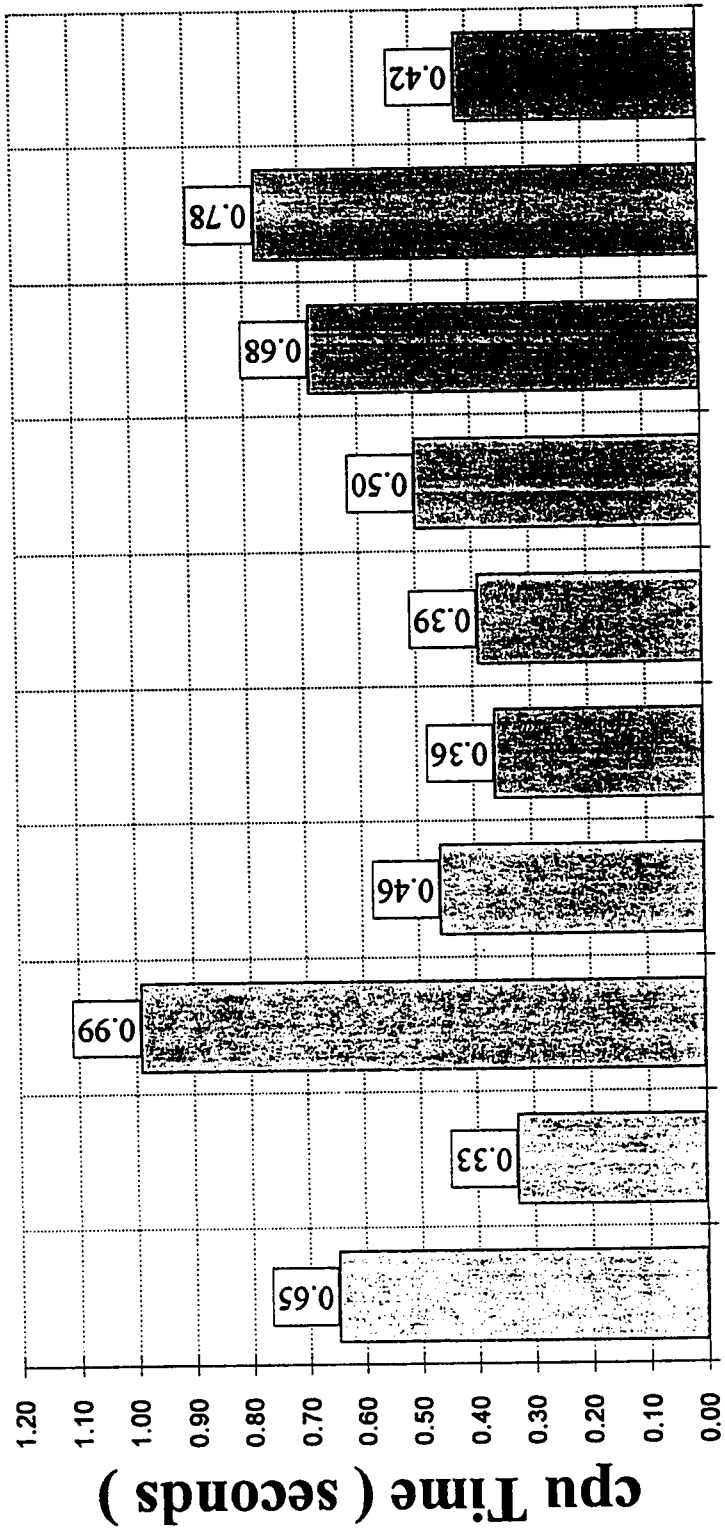



Figure 71 - cpu Time per Step

# CHAPTER 7

# CONCLUSIONS

From this study certain conclusions can be drawn as follows :

1. For the direct solution methods, the cpu time for individual time steps is constant. They will always converge to the right solution after the completion of a certain number of calculations.
2. For the iterative methods, the solution is obtained after a certain number of successive iterations. The iteration is stopped when the convergence criterion is met. Therefore, cpu time for individual time steps will vary according to the number of iterations it takes to converge.
3. For the iterative methods, the number of iterations for a time step was found generally increasing as the time step size increased.
4. A disadvantage of the iterative methods **PSOR, LSOR, ADIP,** and **SIP** is the difficulty in finding the optimum relaxation parameters which are crucial to their performance. Any change in the reservoir problem will require new optimum relaxation parameters to be obtained.
5. The alternating diagonal ordering ( **D4** ) has been found to be more efficient

than the standard ordering.

6. The direct method **D4 - GAUSS** was found to be the most efficient of the direct methods in terms of both the cpu time used and the memory space requirement.
7. **Gaussian Elimination** is a more efficient direct solution procedure than **Matrix De-composition** in terms of both memory space and cpu time requirements.
8. The **RAD** method is a straight forward solution procedure in which computations are primarily performed on the lower right hand quadrant of the matrix, which makes it an efficient solver. It ranked second among the direct methods and was even competitive with some of the iterative methods for the problem at hand in terms of cpu time used.
9. **STBAND** method was the least efficient of the direct methods in terms of both the cpu time and the storage requirement.
10. **PSOR** method was the most efficient of all the iterative as well as the direct methods in terms of the memory space requirement. It was the most efficient in terms of the cpu time requirement among the iterative techniques.
11. **Gaussian Elimination** is a faster solution procedure of tridiagonal matrices than the **Thomas Algorithm.**
12. **LSOR** method took less iterations to converge than **PSOR**. However, since the cpu time per step per iteration for this method was higher than that of **PSOR**, this did not produce any beneficial reduction in total cpu time used. However, the method may converge in certain problems in which **PSOR** fails.
13. **ADIP** method took less iterations to converge than **LSOR** because it iterates on the problem in two directions, once every row and then every column. This caused a faster convergence. However, since the cpu time per step per iteration for this method was higher than that of **LSOR**, it consumed a higher total cpu time than that of **LSOR**.

14. **SIP** method took more iterations to converge than **ADIP.** It was less efficient than **ADIP**, and the **SOR** methods in terms of both memory space and cpu time requirements.
15. **CGTD** and **NF** methods are iterative procedures which do not require any iteration parameters which makes them superior in this respect to the other iterative methods **PSOR, LSOR, ADIP** and **SIP**.
16. **CGTD** method required the calculation of an approximate symmetrical matrix and an approximate inverse of that matrix. The number of iterations this method will take is dependent on how close the approximate inverse of the matrix. This method was the least efficient of the iterative techniques in terms of both the memory space and cpu time requirements.
17. **NF** method took the least number of iterations to converge than the other iterative methods.

# CHAPTER 8

# RECOMMENDATIONS

The comparative evaluation performed in this thesis was limited to two-dimensional reservoir fluid flow and the test problem chosen for this study was a simple homogeneous five-spot water flooding. Therefore, the results obtained and the conclusions arrived at in this study should not be generalized for other types of reservoir problems.

Further evaluation of these methods in solving other types of two-dimensional reservoir problems is recommended. Further evaluation is also recommended in solving three-dimensional reservoir flow equations as well.

# NOMENCLATURE

## Latin Symbols

| | |
|---|---|
| **A** | coefficient matrix |
| **ADIP** | Alternating Direction Implicit Procedure |
| $a_i$, $b_i$, $c_i$, $e_i$, $f_i$ | coefficients of the general pressure equation |
| $a_{ij}$ | an element of the coefficient matrix |
| $A_s$ | symmetric approximation of the matrix A in **CGTD** method |
| A3 | alternating point ordering |
| **B** | pre-conditioning matrix in **NF** method |
| $bx_i$, $by_i$ | the two space derivatives contribution to $b_i$ in **ADIP** |
| **CGTD** | Conjugate Gradient Truncated-Direct Method |
| **d** | right-hand side vector of pressure equation |
| $\Delta P$ | the change in pressure vector |
| $\Delta x$ | length of the grid cell in the x direction |
| $\Delta y$ | length of the grid cell in the y direction |
| **D2** | diagonal ordering |
| **D4** | alternating diagonal ordering |
| $G_{i,j}$, $H_{i,j}$ | vectors in **STBAND** and **RAD** methods |
| **H, V** | the horizontal and vertical matrices in **ADIP** |
| **I** | the identity matrix |
| $k_x$ | permeability in the x direction |
| $k_y$ | permeability in the y direction |

| | |
|---|---|
| $\varphi u_i$ | contribution from the time derivative |
| **L** | lower matrix |
| $L_1$ , $L_2$ | lower bands in NF method |
| $L_d$ | length of the diagonal along which numbering is applied in **RAD** method |
| **LSOR** | Line Successive Over-relaxation method |
| $l_{ij}$ | elements of the lower matrix |
| **M** | diagonal matrix formed in NF method |
| m | semi-bandwidth |
| N | number of cells in the simulation grid |
| $N_d$ | number of diagonals in the system |
| **NF** | Nested Factorization method |
| $N_l$ | number of rows in the lower half of matrix A |
| $N_u$ | number of rows in the upper half of matrix A |
| $O_{i,j}$ | term in **CGTD** method which is contribution of compressibility and a pressure boundry well |
| $\tilde{\mathbf{P}}$ | intermediate calculated pressures in the iterative methods |
| **P** | pressure vector |
| **PSOR** | Point Successive Over-relaxation method |
| **R** | residual vector in **SIP** |
| $r_i$ | residual error |
| $r_o$ | initial residual vector in **CGTD** and NF method |
| S | storage |
| **SIP** | Strongly Implicit Procedure |
| **T** | intermediate matrix in **CGTD** method |
| **U** | upper matrix |
| $u^*$ | intermediate solution to the vector **u** in iterative methods |

| | |
|---|---|
| $U_1$ , $U_2$ | upper bands in NF method |
| $u_{ij}$ | elements of the upper matrix |
| V | intermediate vector in SIP |
| w , v | intermediate vectors in NF method |
| W | work |

## Greek Symbols

| | |
|---|---|
| $\delta$ | the change in pressures vector in SIP |
| $\varepsilon$ | maximum absolute tolerance |
| $\lambda$ | transmissibility and is equal to $k/(\mu B)$ |
| $\eta$ | $N_x$ - the number of cells in the x direction |
| $\nu$ | iteration number |
| $\zeta$ , $\xi$ | vectors in Thomas Algorithm |
| $\omega$ | iteration relaxation parameter |
| $\omega_{opt}$ | optimum iteration relaxation parameter |

# REFERENCES


1. Coats, K. H. and Terhune, M. H., " Comparison of Alternating Direction Explicit and Implicit Procedures in Two-Dimensional Flow Calculations, " SPEJ, ( 1966 ), pp. 350 - 62, Trans. AIME, Vol. 237.
2. Stone, H. L., " Iterative Solution of Implicit Approximations of Multi-Dimensional Partial Differential Equations, " SIAM J. Numer. Anal. ( Sept. 1968 ), Vol. 5, No. 3, p. 630.
3. Breitenbach, E. A., Thurnau, D. H. and Van Pollen, H. K., " Solution of the Immiscible Fluid Flow Simulation Equations, " Paper SPE 2021 presented at 1st Numerical Simulation Symposium, Dallas, Texas, ( April, 1968 ).
4. Briggs, J. E. and Dixon, T. N., " Some Practical Considerations in the Numerical Solution of Two-Dimensional Reservoir Problems, " SPEJ, ( June, 1968 ) pp. 185 - 194.
5. Bjordammen, J. B. and Coats, K. H., " Comparison of Alternating Direction and Successive Over-relaxation Techniques in Simulation of Reservoir Fluid Flow, " SPEJ, ( 1969 ), Vol. 9, No. 1, pp. 47 - 58.
6. Weinstein, H. G., Stone, H. L., and Kwan, T. V., " Simultaneous Solution of Multiphase Reservoir Flow Equations, " SPE Reprint No. 11, ( 1973 ), pp. 124 -135.
7. Watts, J. W., " An Iterative Matrix Solution Method Suitable for Anisotropic Problems, " SPE Reprint No. 11, ( 1973 ), pp. 119 - 123.

8. Traylor, C. and Sheffield, M., " The Comparable Efficiency of Mathematical Techniques Used in Reservoir Simulation, " Paper SPE 3560 presented at the 46th Annual Fall Meeting, New Oreleans, ( 1971 ).

9. Price, H. S. and Coats, K. H., " Direct Methods in Reservoir Simulation," SPEJ, ( June, 1974 ), No. 14, pp. 295 - 308.

10. Al-Marhoun, M. A., " A New Approach to the Application of Direct Methods in Petroleum Reservoir Simulation, " The Arabian Journal for Science and Engineering , Vol. 7, No. 3, pp. 183 - 190.

11. Watts, J. W., " A Conjugate Gradient-Truncated Direct Method for the Iterative Solution of the Reservoir Simulation Pressure Equation, " SPEJ, ( June, 1981 ), pp. 345 - 353.

12. Appleyard, J. R. and Cheshire, I. M., " Nested Factorization, " Paper SPE 12264 presented at the SPE Symposium on Reservoir Simulation, San Francisco, ( Nov. 1974 ).

13. Settari, A. and Aziz, K., " A Generalization of the Additive Correction Methods for the Iterative Solution of Matrix Equations, " SIAM J. Numer. Anal., ( June, 1973 ), 10, 506 - 521.

14. Harouaka, A.: A New Block Sparse Direct Solution Technique - Application to Hydrocarbon Reservoir Simulation, Ph.D. Dissertation, Penn. State Univ., Pennsylvania, ( 1987 ).

15. Odeh, A. S., " Comparison of Solutions to a Three-Dimensional Black-Oil Reservoir Simulation Problem," Journal of Petroleum Technology, ( January, 1981 ), 13 - 25.

16. Crichlow, H. B.: "Modern Reservoir Engineering - A Simulation Approach, " Englewood Cliffs, New Jersy : Prentice Hall, 1977.

17. Woo, P. F., Roberts, S. J. and Gustavson, F. G., "Application of Sparse Matrix Techniques in Reservoir Simulation, " SPE 4544, paper presented at the 48th Annual Meeting, Las Vegas, Nev., Oct. 1973.

18. Meijerink, J. A. and Van der Vorst, H. A., " An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix, " Math. of Compt., 31, No. 137, pp. 148 - 62.

19. Aziz K. and Settari, A. : Petroleum Reservoir Simulation, Applied Science Publishers Ltd, London ( 1979 ).

20. Appleyard, J. R., Cheshire, I. M., and Pollard, R. K., " Special Techniques for Fully Implicit Simulators, " Proc. European Symposium on Enhanced Oil Recovery, Bournemouth, England ( 1981 ), 395 - 408.

21. Burden R., Faires J., and Reynolds A.: Numerical Analysis, Prindle, Weber and Schmidt, Boston ( 1981 ).

22. Al-Marhoun, M. A.: Optimal Numerical Procedure to Solve Two-dimensional Three-phase Petroleum Reservoir Simulator, Ph.D. Dissertation, Norman, Oklahoma, 1978.

23. Al-Yousef, H. Y.: An Improved Method to Calculate Pseudorelative Permeabilities to Model Gravity and Capillary Effects in Reservoir Simulation, Ph. D. Dissertation, Stanford University, California, June 1985.

# APPENDIX
# LISTING OF COMPUTER PROGRAMS

```
C      THE STANDARD - GAUSS METHOD
C
       SUBROUTINE GBAND3(A,B,C,E,F,D,CALP,NX,NY,NXNY)
       REAL*8     E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),
      *           D(NXNY),CALP(NXNY),
      *           AA(361,-19:19),SUM,VAL
       INTEGER*4 NX,NY,NXNY
       DATA NFLAG/0/
       IF(NFLAG.EQ.1)GO TO 48
       NFLAG=1
       NXM1=NX-1
       NXNYM1=NXNY-1
       NXNYNXM1=NXNY-NXM1
       NXNYMNX=NXNY-NX
       DO 23 N=1,NXNY
       DO 23 K=-NX,NX
23     AA(N,K)=0
       GO TO 49
48     DO 24 N=1,NXNY
       DO 24 K=2,NXM1
24     AA(N,K)=0
49     DO 999 N=1,NXNY
       AA(N,-NX)=E(N)
       AA(N,-1)=A(N)
       AA(N,0)=B(N)
       AA(N,1)=C(N)
       AA(N,NX)=F(N)
       CALP(N)=D(N)
999    CONTINUE
       DO 51 K=1,NXNYMNX
       DO 47 I=K+1,K+NX
       KS=K-I
       IF(AA(I,KS).EQ.0)GO TO 47
       VAL=AA(I,KS)/AA(K,0)
       DO 11 N=KS,KS+NX
       AA(I,N)=AA(I,N)-AA(K,N-KS)*VAL
11     CONTINUE
       CALP(I)=CALP(I)-CALP(K)*VAL
47     CONTINUE
51     CONTINUE
       DO 510 K=NXNYNXM1,NXNYM1
       NMX=NXNY-K
       DO 470 I=K+1,NXNY
       KS=K-I
       IF(AA(I,KS).EQ.0)GO TO 470
       VAL=AA(I,KS)/AA(K,0)
       DO 31 N=KS,KS+NMX
       AA(I,N)=AA(I,N)-AA(K,N-KS)*VAL
31     CONTINUE
       CALP(I)=CALP(I)-CALP(K)*VAL
```

```
470   CONTINUE
510   CONTINUE
      CALP(NXNY)=CALP(NXNY)/AA(NXNY,0)
      DO 120 I=NXNYM1,NXNYNXM1,-1
      JE=NXNY-I
      DO 122 J=1,JE,1
122   CALP(I)=CALP(I)-AA(I,J)*CALP(I+J)
120   CALP(I)=CALP(I)/AA(I,0)
      DO 1230 I=NXNYMNX,1,-1
      DO 1223 J=1,NX,1
1223  CALP(I)=CALP(I)-AA(I,J)*CALP(I+J)
1230  CALP(I)=CALP(I)/AA(I,0)
      RETURN
      END
```

```
C      THE D4 - GAUSS METHOD
C
       SUBROUTINE D4(A,B,C,D,E,F,PCAL,NX,NY,NXNY)
       REAL*8 A(NXNY),B(NXNY),C(NXNY),E(NXNY),F(NXNY),D(NXNY),
      *  PCAL(NXNY),AA(182:361,-19:19),SUM,VAL,CALP(361),AL1(181),
      *    AU1(181,4),AL2(181,4)
       INTEGER*4 NX,NY,NACT,INDEX(361),KEY(0:20,0:20),L(361,4),
      *    NFLAG,IND(181,4),IND2(181),INDC(181,4),IND3(181,4),
      *          INDC3(181,4),IND33(181)
       DATA      KEY,NFLAG/441*0,1*0/
       IF(NFLAG.EQ.1) GO TO 223
       NFLAG=1
C REORDER THE GRID
       NACT=0
       DO 281 N=1,NX,2
       I=N
       JLAST=N
       IF(JLAST.GT.NY) JLAST=NY
       DO 281  J=1,JLAST,1
       NN=I+(J-1)*NX
       NACT=NACT+1
       INDEX(NACT)=NN
       KEY(I,J)=NACT
281    I=I-1
3331   N=N-2
       IDELX=NX-N
       IF(IDELX.EQ.1) JSTART=2
       IF(IDELX.EQ.0) JSTART=3
461    I=NX
       JLAST=JSTART+NX-1
       IF(JLAST.GT.NY) JLAST=NY
       DO 351 J=JSTART,JLAST,1
       NN=I+(J-1)*NX
       NACT=NACT+1
       INDEX(NACT)=NN
       KEY(I,J)=NACT
       IF(KEY(I,J).GE.NXNY)GO TO 221
351    I=I-1
       JSTART=JSTART+2
       IF(JSTART.LE.NY) GO TO 461
       DO 2011 N=2,NX,2
       I=N
       JLAST=N
       IF(JLAST.GT.NY) JLAST=NY
       DO 2011 J=1,JLAST,1
       NN=I+(J-1)*NX
       NACT=NACT+1
       INDEX(NACT)=NN
       KEY(I,J)=NACT
2011   I=I-1
```

```
      GO TO  3331
221   NAC=(NACT-1)/2 +1
      NACTP1=NACT+1
      NACP1=NAC+1
      DO 67 JJ=1,NY
      DO 67 II=1,NX
      M=KEY(II,JJ)
      L(M,1)=KEY(II-1,JJ)
      L(M,2)=KEY(II+1,JJ)
      L(M,3)=KEY(II,JJ-1)
67    L(M,4)=KEY(II,JJ+1)
      DO 993 K=1,NAC
      LL=0
      N=INDEX(K)
      DO 992 J=1,4
      IF(L(K,J).NE.0) THEN
      LL=LL+1
      M=L(K,J)
      IF(J.EQ.1)AU1(K,LL)=A(N)
      IF(J.EQ.2)AU1(K,LL)=C(N)
      IF(J.EQ.3)AU1(K,LL)=E(N)
      IF(J.EQ.4)AU1(K,LL)=F(N)
      IND3(K,LL)=M
      INDC3(K,LL)=J
      ENDIF
992   CONTINUE
      IND33(K)=LL
993   CONTINUE
      DO 996 K=1,NAC
      N=0
      DO 997 M=NACP1,NACT
      NN=INDEX(M)
      IF(L(M,1).EQ.K) THEN
      N=N+1
      AL2(K,N)=A(NN)
      IND(K,N)=M
      INDC(K,N)=1
      ENDIF
      IF(L(M,2).EQ.K) THEN
      N=N+1
      AL2(K,N)=C(NN)
      IND(K,N)=M
      INDC(K,N)=2
      ENDIF
      IF(L(M,3).EQ.K) THEN
      N=N+1
      AL2(K,N)=E(NN)
      IND(K,N)=M
      INDC(K,N)=3
      ENDIF
      IF(L(M,4).EQ.K) THEN
      N=N+1
      AL2(K,N)=F(NN)
      IND(K,N)=M
```

```
      INDC(K,N)=4
      ENDIF
997   CONTINUE
      IND2(K)=N
996   CONTINUE
      DO 998 M=1,NAC
      N=INDEX(M)
      AL1(M)=B(N)
998   CALP(M)=D(N)
      GO TO 48
223   DO 991 K=1,NAC
      KE=IND33(K)
      N=INDEX(K)
      AL1(K)=B(N)
      CALP(K)=D(N)
      DO 991 J=1,KE
      IF(INDC3(K,J).EQ.1)AU1(K,J)=A(N)
      IF(INDC3(K,J).EQ.2)AU1(K,J)=C(N)
      IF(INDC3(K,J).EQ.3)AU1(K,J)=E(N)
991   IF(INDC3(K,J).EQ.4)AU1(K,J)=F(N)
      DO 981 K=1,NAC
      KE=IND2(K)
      DO 981 J=1,KE
      M=IND(K,J)
      N=INDEX(M)
      IF(INDC(K,J).EQ.1)AL2(K,J)=A(N)
      IF(INDC(K,J).EQ.2)AL2(K,J)=C(N)
      IF(INDC(K,J).EQ.3)AL2(K,J)=E(N)
981   IF(INDC(K,J).EQ.4)AL2(K,J)=F(N)
48    DO 116 N=NACP1,NACT
      DO 116 K=-NX,NX
116   AA(N,K)=0.0
      DO 999 M=NACP1,NACT
      N=INDEX(M)
      AA(M,0)=B(N)
999   CALP(M)=D(N)
C *****SOLVE FOR PRESSURES ***************
      DO 51 K=1,NAC
      KE1=IND33(K)
      KE2=IND2(K)
      DO 57 M=1,KE2
      I=IND(K,M)
      VAL=AL2(K,M)/AL1(K)
      DO 34 NM=1,KE1
      KK=IND3(K,NM)
      IK=KK-I
34    AA(I,IK)=AA(I,IK)-AU1(K,NM)*VAL
      CALP(I)=CALP(I)-CALP(K)*VAL
57    CONTINUE
51    CONTINUE
C*************************************************
      DO 11 K=NACP1,NACT-NX
      DO 13 I=K+1,K+NX
      KS=K-I
```

```
      KE=KS+NX
      IF(AA(I,KS).EQ.0)GO TO 13
      VAL=AA(I,KS)/AA(K,0)
      DO 12 J=KS+1,KE
12    AA(I,J)=AA(I,J)-AA(K,J-KS)*VAL
      CALP(I)=CALP(I)-CALP(K)*VAL
13    CONTINUE
11    CONTINUE
      NXM1=NX-1
      DO 21 K=NACT-NXM1,NACT-1
      DO 23 I=K+1,NACT
      KS=K-I
      KE=KS+NX
      IF(AA(I,KS).EQ.0)GO TO 23
      VAL=AA(I,KS)/AA(K,0)
      DO 22 J=KS+1,KE
22    AA(I,J)=AA(I,J)-AA(K,J-KS)*VAL
      CALP(I)=CALP(I)-CALP(K)*VAL
23    CONTINUE
21    CONTINUE
C**** DO BACKWARD SUBSTITUTION **********************
      CALP(NACT)=CALP(NACT)/AA(NACT,0)
      DO 140 I=NACT-1,NACT-NX,-1
      DO 142 J=I+1,NACT,1
      CALP(I)=CALP(I)-AA(I,J-I)*CALP(J)
142   CONTINUE
      CALP(I)=CALP(I)/AA(I,0)
140   CONTINUE
      NXP1=NX+1
      DO 130 I=NACT-NXP1,NACP1,-1
      DO 132 J=1,NX,1
      IPJ=I+J
      CALP(I)=CALP(I)-AA(I,J)*CALP(IPJ)
132   CONTINUE
      CALP(I)=CALP(I)/AA(I,0)
130   CONTINUE
      DO 1120 I=NAC,1,-1
      N=INDEX(I)
      KE=IND33(I)
      DO 33 J=1,KE
      M=IND3(I,J)
33    CALP(I)=CALP(I)-AU1(I,J)*CALP(M)
      PCAL(N)=CALP(I)/AL1(I)
1120  CONTINUE
      DO 10 M=NACP1,NACT
      N=INDEX(M)
10    PCAL(N)=CALP(M)
      RETURN
      END
```

```
C      THE STBAND METHOD
C
       SUBROUTINE STBND(A,B,C,E,F,D,CALP,NX,NY,NXNY)
       REAL*8     E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),
      *           D(NXNY),CALP(380),
      *           Y(-18:380,19),BETA(361),ALFA(361,19),
      *           GAMMA(-18:380),G(361,19),H(361,19),SUM
       INTEGER*4 NX,NY,NXNY
       IF(NFLAG.EQ.1)GO TO 12
       NFLAG=1
       DO 11 N=1-NX,NXNY+NX
       GAMMA(N)=0
       DO 11 K=1,NX
11     Y(N,K)=0
       DO 15 N=1,NXNY
       DO 15 K=1,NX
15     H(N,K)=0
12     DO 23 N=1,NXNY
       DO 23 I=2,NX-1
       ALFA(N,I)=0.0
23     CONTINUE
       DO 37 N=1,NXNY
       ALFA(N,1)=A(N)
       ALFA(N,NX)=E(N)
       DO 39 I=NX,1,-1
       DO 41 K=I+1,NX,1
       ALFA(N,I)=ALFA(N,I)-ALFA(N,K)*Y(N-K,K-I)
41     CONTINUE
39     CONTINUE
       BETA(N)=B(N)
       DO 42 K=1,NX,1
       NMK=N-K
       BETA(N)=BETA(N)-ALFA(N,K)*Y(NMK,K)
42     CONTINUE
       H(N,1)=C(N)
       H(N,NX)=F(N)
       DO 59 I=1,NX,1
       SUM=H(N,I)
       NPI=N+I
       DO 44 K=I+1,NX,1
       KMI=K-I
       NMKPI=NPI-K
       SUM=SUM-ALFA(N,KMI)*Y(NMKPI,K)
44     CONTINUE
       Y(N,I)=(1/BETA(N))*SUM
       SUM=D(N)
       DO 45 K=1,NX,1
       SUM=SUM-ALFA(N,K)*GAMMA(N-K)
45     CONTINUE
       GAMMA(N)=(1/BETA(N))*SUM
```

```
59      CONTINUE
37      CONTINUE
        DO 47 N=NXNY,1,-1
        CALP(N)=GAMMA(N)
        DO 46 K=1,NX,1
        CALP(N)=CALP(N)-Y(N,K)*CALP(N+K)
46      CONTINUE
47      CONTINUE
        RETURN
        END
```

```
C      THE RADF METHOD
C
       SUBROUTINE RADP(A,B,C,E,F,D,CALP,NX,NY,NXNY)
       REAL*8     E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),
      *           D(NXNY),CALP(NXNY),Y(0:722),H(21,380),G(21,380)
       INTEGER*4 NX,NY,NUDX(190),NLDX(190),KEY(722)
       DATA IFLAG2/0/
       IF(IFLAG2.EQ.1)GO TO 12
       NACT=0
       JS=0
       NN=0
       DO 381 IS=19,1,-1
       IE=IS +18
       JS=JS+1
       J=JS
       DO 281 I=IS,IE
       NACT=NACT+1
       IF(I.GT.9.AND.I.LT.29.AND.J.GT.9.AND.J.LT.29) THEN
       NN=NN+1
       NUDX(NN)=NACT
       NEW=(I-9)+(J-10)*NX
       KEY(NACT)=NEW
       ENDIF
       J=J+1
 281   CONTINUE
 381   CONTINUE
       NUD=NN
       JS=1
       DO 481 IS=19,1,-1
       IE=IS +18
       JS=JS+1
       J=JS
       DO 581 I=IS,IE
       NACT=NACT+1
       IF(I.GT.9.AND.I.LT.29.AND.J.GT.9.AND.J.LT.29) THEN
       NN=NN+1
       NLDX(NN-NUD)=NACT
       NEW=(I-9)+(J-10)*NX
       KEY(NACT)=NEW
       ENDIF
       J=J+1
 581   CONTINUE
 481   CONTINUE
       NLD=NXNY-NUD
       NU=NACT/2
       NL=NACT-NU
       M=NX+2
       M1=M-1
       M2=M-2
 12    DO 37 K=1,NUD
```

```
      J=NUDX(K)
      N=KEY(J)
      A(N)=A(N)/B(N)
      E(N)=E(N)/B(N)
      C(N)=C(N)/B(N)
      F(N)=F(N)/B(N)
37    Y(J)=D(N)/B(N)
      DO 38 JN=1,NLD
      K=NLDX(JN)
      J=K-NU
      JJ=KEY(J)
      KK=KEY(K)
      J1=KEY(J+1)
      K1=KEY(J+M1)
      K2=KEY(J+M2)
      G(1,J)=-E(KK)*E(JJ)
      G(2,J)=-E(KK)*C(JJ)-C(KK)*E(J1)
      G(3,J)=-C(KK)*C(J1)
      G(M1,J)=-E(KK)*A(JJ)-A(KK)*E(K2)
      G(M,J)=B(KK)-E(KK)*F(JJ)-C(KK)*A(J1)
     *       -A(KK)*C(K2)-F(KK)*E(K1)
      H(1,J)=-F(K1)*F(KK)
      H(2,J)=-F(K2)*A(KK)-A(K1)*F(KK)
      H(3,J)=-A(K2)*A(KK)
      H(M1,J)=-F(J1)*C(KK)-C(K1)*F(KK)
      Y(K)=D(KK)
      DO 39 I=4,M2
      G(I,J)=0.0
      H(I,J)=0.0
39    CONTINUE
38    CONTINUE
C GROUP 1   ***************************
C ************************************
      ISS=NLDX(1)-NU
      DO 41 I=1,M
41    H(I,ISS)=H(I,ISS)/G(M,ISS)
      DO 40 JJ=2,2
      J=NLDX(JJ)-NU
      J0=J-1
      J2=J+1
      G(M,J)=G(M,J)-G(M1,J)*H(M1,J0)
      H(1,J)=H(1,J)/G(M,J)
      DO 42 I=2,J2
42    H(I,J)=(1/G(M,J))*(H(I,J)-H(I-1,J0)*G(M1,J))
40    H(M1,J)=H(M1,J)/G(M,J)
C   GROUP  2
C*********************************************************
      DO 43 JJ=3,NLD
      J=NLDX(JJ)-NU
      JM=J-M
      DO 44 I=1,M
      JMI=JM+I
      KE=I-1
      DO 44 K=1,KE
```

```
44     G(I,J)=G(I,J)-G(I-K,J)*H(M-K,JMI-K)
       H(1,J)=H(1,J)/G(M,J)
       DO 46 I=2,M1
       KE=I-1
       DO 47 K=1,KE
47     H(I,J)=H(I,J)-H(I-K,J-K)*G(M-K,J)
46     H(I,J)=H(I,J)/G(M,J)
43     CONTINUE
C      BACK SUBSTITUTION
C*********************************************
       DO 50 K=1,NLD
       J=NLDX(K)
       JJ=KEY(J)
       JNU=J-NU
       JM=J-M
       Y(J)=Y(J)-E(JJ)*Y(JNU)-C(JJ)*Y(JNU+1)
      *         -A(JJ)*Y(JNU+M2)-F(JJ)*Y(JNU+M1)
       DO 61 KK=1,M1
61     Y(J)=Y(J)-G(KK,JNU)*Y(JM+KK)
50     Y(J)=Y(J)/G(M,JNU)
       DO 52 JJ=NLD-1,1,-1
       J=NLDX(JJ)
       JNU=J-NU
       JM=J+M
       DO 52 K=1,M1
52     Y(J)=Y(J)-H(K,JNU)*Y(JM-K)
       DO 56 K=NUD,1,-1
       J=NUDX(K)
       JJ=KEY(J)
       JNU=J+NU
       Y(J)=Y(J)-F(JJ)*Y(JNU)-A(JJ)*Y(JNU-1)
      *         -C(JJ)*Y(JNU-M2)-E(JJ)*Y(JNU-M1)
56     CALP(JJ)=Y(J)
       DO 671 J=1,NLD
       L=NLDX(J)
       N=KEY(L)
671    CALP(N)=Y(L)
       RETURN
       END
```

```
C      THE PSOR METHOD
C
       SUBROUTINE PSOR(A,B,C,E,F,D,PE,CALP,NX,NY,NXNY)
       REAL*8  E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),ERROR,
      *       PE(NXNY),D(NXNY),R,CALP(NXNY),PSAVE(441),W,RMAX
       INTEGER*4 NX,NY,NXNY,INDEX(361)
       IF(NFLAG.EQ.1)GO TO 122
       NFLAG=1
       ERROR=0.0001
       DO 77 N=1,NXNY
       IF(N.LE.NX)M=N+NX+3
       IF(N.GT.NX)M=N+NX+3+2*J
       J=N/NX
       INDEX(N)=M
77     CONTINUE
       NXP2=NX+2
       W=1.790
       NTRIAL=500
12     DO 67 M=1,441
67     PSAVE(M)=0.0
122    DO 32 K=1,NTRIAL,1
       RMAX=0.0
       DO 34 N=1,NXNY
       M=INDEX(N)
       CALP(N)=(D(N)-E(N)*PSAVE(M-NXP2)
      *              -A(N)*PSAVE(M-1)
      *              -C(N)*PSAVE(M+1)
      *              -F(N)*PSAVE(M+NXP2))/B(N)
       CALP(N)=PSAVE(M)+W*(CALP(N)-PSAVE(M))
       R=ABS((CALP(N)-PSAVE(M))/CALP(N))
       IF(R.GT.RMAX)RMAX=R
       PSAVE(M)=CALP(N)
34     CONTINUE
       IF(RMAX.LE.ERROR) GO TO 31
32     CONTINUE
31     RETURN
       END
```

```
C      THE LSOR METHOD
C
       SUBROUTINE LSOR(A,B,C,E,F,D,PE,CALP,NX,NY,NXNY,NUM)
       REAL*8 E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),RES,BB(361),
      *       D2(361),PE(NXNY),D(NXNY),CALP(NXNY),PSAVE(441),VAL
       INTEGER*4 NX,NY,NXNY,INDEX(361),KEY(19,19)
       DATA PSAVE,NFLAG/441*0,0/
       IF(NFLAG.EQ.1)GO TO  122
       NFLAG=1
       ERROR=0.0001
       W=1.710
       NTRIAL=500
       NXP3=NX+3
       DO 77 N=1,NXNY
       IF(N.LE.NX)M=N+NXP3
       IF(N.GT.NX)M=N+NXP3+2*J
       J=N/NX
       INDEX(N)=M
77     CONTINUE
       NXP2=NX+2
       NXM1=NX-1
       NXM2=NX-2
122    DO 32 K=1,NTRIAL,1
       RMAX=0.0
       L=0
       DO 34 J=1,NY
       DO 90 I=1,NX
       L=L+1
       N=INDEX(L)
       D2(L)=D(L)-E(L)*PSAVE(N-NXP2)
      *          -F(L)*PSAVE(N+NXP2)
       BB(L)=B(L)
90     CONTINUE
       DO 37 I=L-NXM2,L
       IM1=I-1
       VAL=A(I)/BB(IM1)
       BB(I)=BB(I)-VAL*C(IM1)
       D2(I)=D2(I)-VAL*D2(IM1)
37     CONTINUE
       CALP(L)=D2(L)/BB(L)
       DO 38 I=L-1,L-NXM1,-1
       CALP(I)=(D2(I)-C(I)*CALP(I+1))/BB(I)
38     CONTINUE
       DO 15 I=L-NXM1,L
       N=INDEX(I)
       CALP(I)=PSAVE(N)+W*(CALP(I)-PSAVE(N))
       RES=ABS((CALP(I)-PSAVE(N))/CALP(I))
       IF(RES.GT.RMAX)RMAX=RES
       PSAVE(N)=CALP(I)
15     CONTINUE
```

```
34      CONTINUE
        IF(RMAX.LE.ERROR) GO TO 41
32      CONTINUE
41      RETURN
        END
```

```
C     THE ADIP METHOD
C
      SUBROUTINE ADIP(A,B,C,E,F,D,CALP,NX,NY,NXNY)
      REAL*8     E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),VAL,
     *           D(NXNY),PSAVE(441),RES,BB(361),
     *           CALP(NXNY),D2(361)
      INTEGER*4 NX,NY,NXNY,INDEX(361)
      DATA PSAVE,NFLAG/441*0,0/
      IF(NFLAG.EQ.1)GO TO 122
      NFLAG=1
      ERROR=.0001
      W=1.770
      NTRIAL=500
      NXP3=NX+3
      DO 77 N=1,NXNY
      IF(N.LE.NX)M=N+NXP3
      IF(N.GT.NX)M=N+NXP3+2*J
      J=N/NX
      INDEX(N)=M
77    CONTINUE
      NXP2=NX+2
      NXM2=NX-2
      NXM1=NX-1
      DO 5 K=1,NTRIAL
      RMAX=0.0
C DO HORIZONTAL SWEEP    ****************************************
      L=0
      DO 34 J=1,NY,1
      DO 35 I=1,NX,1
      L=L+1
      N=INDEX(L)
      D2(L)=D(L)-E(L)*PSAVE(N-NXP2)-F(L)*PSAVE(N+NXP2)
      BB(L)=B(L)
35    CONTINUE
      DO 37 I=L-NXM2,L
      IM1=I-1
      VAL=A(I)/BB(IM1)
      BB(I)=BB(I)-VAL*C(IM1)
      D2(I)=D2(I)-VAL*D2(IM1)
37    CONTINUE
      CALP(L)=D2(L)/BB(L)
      N=INDEX(L)
      PSAVE(N)=CALP(L)
      DO 38 I=L-1,L-NXM1,-1
      N=INDEX(I)
      CALP(I)=(D2(I)-C(I)*CALP(I+1))/BB(I)
      PSAVE(N)=CALP(I)
38    CONTINUE
34    CONTINUE
C DO VERTICAL SWEEP    ****************************************
```

```
      DO 341 I=1,NX,1
      DO 351 J=1,NY,1
      M=I+(J-1)*NX
      N=INDEX(M)
      BB(M)=B(M)
351   D2(M)=D(M)-A(M)*PSAVE(N-1)-C(M)*PSAVE(N+1)
      DO 371 IS=I+NX,M,NX
      IM1=IS-NX
      VAL=E(IS)/BB(IM1)
      BB(IS)=BB(IS)-VAL*F(IM1)
      D2(IS)=D2(IS)-VAL*D2(IM1)
371   CONTINUE
      CALP(M)=D2(M)/BB(M)
      DO 381 II=M-NX,I,-NX
      CALP(II)=(D2(II)-F(II)*CALP(II+NX))/BB(II)
381   CONTINUE
      DO 152 J=I,M,NX
      N=INDEX(J)
      CALP(J)=PSAVE(N)+W*(CALP(J)-PSAVE(N))
      RES=ABS((CALP(J)-PSAVE(N))/CALP(J))
      IF(RES.GT.RMAX)RMAX=RES
      PSAVE(N)=CALP(J)
152   CONTINUE
341   CONTINUE
      IF(RMAX.LE.ERROR) GO TO 31
5     CONTINUE
31    RETURN
      END
```

```
C      THE SIP METHOD
C
       SUBROUTINE SIP(A,B,C,E,F,D,CALP,NX,NY,NXNY,NUM)
       REAL*8 D(NXNY),V(441),
      *       CALP(NXNY),E(NXNY),A(NXNY),B(NXNY),C(NXNY),F(NXNY),
      *       EE(441),AA(441),BB(441),CC(441),FF(441),RES,
      *       PSAVE(441),VAL1,VAL2,VAL3,VAL4
       INTEGER*4 NX,NY,NXNY,INDEX(361)
       DATA PSAVE,V,CC,BB,FF,AA,EE,NFLAG/441*0,441*0,441*0,441*0,
      *          441*0,441*0,441*0,0/
       IF(NFLAG.EQ.1)GO TO 121
       NFLAG=1
       NXP3=NX+3
       N=0
       DO 588 J=1,NY
       DO 588 I=1,NX
       N=N+1
       IF(N.LE.NX)M=N+NXP3
       IF(N.GT.NX)M=N+NXP3+2*JJ
       JJ=N/NX
588    INDEX(N)=M
       NTRIAL=150
       ALFA=0.9850
       ERROR=0.0001
       NXP2=NX+2
121    DO 5 K=1,NTRIAL
       RMAX=0
       DO 106 N=1,NXNY
       M=INDEX(N)
       MMNXP2=M-NXP2
       MM1=M-1
       VAL3=ALFA*CC(MMNXP2)
       VAL4=ALFA*FF(MM1)
       EE(M)=E(N)/(1+VAL3)
       AA(M)=A(N)/(1+VAL4)
       VAL1=EE(M)*VAL3
       VAL2=AA(M)*VAL4
       BB(M)=B(N)+VAL1+VAL2
      *      -EE(M)*FF(MMNXP2)
      *      -AA(M)*CC(MM1)
       CC(M)=(C(N)-VAL1)/BB(M)
       FF(M)=(F(N)-VAL2)/BB(M)
       V(M)=(D(N)-E(N)*PSAVE(MMNXP2)
      *          -A(N)*PSAVE(MM1 )
      *          -B(N)*PSAVE(M)
      *          -C(N)*PSAVE(M+1 )
      *          -F(N)*PSAVE(M+NXP2)
      *          -AA(M)*V(MM1)-EE(M)*V(MMNXP2))/BB(M)
106    CONTINUE
       DO 108 N=NXNY,1,-1
```

```
      M=INDEX(N)
      CALP(N)=PSAVE(M)+V(M)-CC(M)*V(M+1)-FF(M)*V(M+NXP2)
      RES=ABS((CALP(N)-PSAVE(M))/CALP(N))
      IF(RES.GT.RMAX)RMAX=RES
      PSAVE(M)=CALP(N)
108   CONTINUE
      IF(RMAX.LE.ERROR) GO TO 339
5     CONTINUE
339   RETURN
      END
```

```
C
C  .   THE CGTD METHOD
C
      SUBROUTINE CGTD(A,B,C,E,F,D,CALP,NX,NY,NXNY,NUM)
      REAL*8  AAS(361,0:4),D(NXNY),PSAVE(361),PSAVE2(361),
     *A(NXNY),B(NXNY),C(NXNY),E(NXNY),F(NXNY),AUINR(361),
     *AUINV(361,0:1),SUM3,AU(361,0:19),ALFSUM(361),ALINR(380),
     *VAL,SSOLD,ALFA,BETA,SUM1,SUM2,SUM,CALP(NXNY),SS(361),
      INTEGER*4   NX,NY,NXNY,KEY(0:20,0:20),NACT,
     *        LOC(361,4),INDEX(361),R(361)
      DATA KEY,LOC/441*0,1444*0/
      IF(NFLAG.EQ.1)GO TO 10
      NFLAG=1
      NACT=0
      DO 381 N=1,NX
      I=N
      JLAST=N
      IF(JLAST.GT.NY) JLAST=NY
      DO 381  J=1,JLAST,1
      NN=I+(J-1)*NX
      NACT=NACT+1
      INDEX(NACT)=NN
      KEY(I,J)=NACT
381   I=I-1
      JSTART=2
461   JLAST=JSTART+NX-1
      I=NX
      IF(JLAST.GT.NY) JLAST=NY
      DO 351 J=JSTART,JLAST,1
      NN=I+(J-1)*NX
      NACT=NACT+1
      INDEX(NACT)=NN
      KEY(I,J)=NACT
351   I=I-1
      JSTART=JSTART+1
      IF(JSTART.LE.NY) GO TO 461
C*********************************
      DO 999 J=1,NY
      DO 999 I=1,NX
      M=KEY(I,J)
      M1=KEY(I,J-1)
      M2=KEY(I-1,J)
      M3=KEY(I+1,J)
      M4=KEY(I,J+1)
      IF(M1.NE.0)
     *LOC(M,1)=M1-M
      IF(M2.NE.0)
     *LOC(M,2)=M2-M
      IF(M3.NE.0)
     *LOC(M,3)=M3-M
```

```
      IF(M4.NE.0)
     *LOC(M,4)=M4-M
999   CONTINUE
C**********************
      DO 989 M=1,NACT
      DO 996 L=0,4
996   AAS(M,L)=0.0
989   CONTINUE
      DO 997 M=1,NACT
      DO 997 L=0,NX
997   AU(M,L)=0.0
10    DO 991 M=1,NACT
      AUINV(M,1)=0.0
      PSAVE(M)=0.0
      PSAVE2(M)=0.0
      N=INDEX(M)
991   R(M)=D(N)
C**********************
      DO 434 N=1,NACT
      M3=LOC(N,3)
      M4=LOC(N,4)
      M=INDEX(N)
      IF(M3.NE.0) THEN
      NPM3=N+M3
      MPM3=INDEX(NPM3)
      AAS(N,3)=(C(M)+A(MPM3))/2
      AAS(NPM3,2)=AAS(N,3)
      ENDIF
      IF(M4.NE.0) THEN
      NPM4=N+M4
      MPM4=INDEX(NPM4)
      AAS(N,4)=(F(M)+E(MPM4))/2
      AAS(NPM4,1)=AAS(N,4)
      ENDIF
      AAS(N,0)=AAS(N,1)+AAS(N,2)+AAS(N,3)+AAS(N,4)+
     *          B(M)-E(M)-A(M)-C(M)-F(M)
434   CONTINUE
C*************************************************
      DO 91 K=1,NACT
      IS=K-NX
      IF(IS.LT.1)IS=1
      SUM=0
      DO 97 I=IS,K-1
97    SUM=SUM+(AU(I,K-I)**2)/AU(I,0)
      AU(K,0)=AAS(K,0)-SUM
      DO 94 J=3,4
      IF(AAS(K,J).EQ.0)GO TO 94
      JJ=LOC(K,J)
      KPJ=K+JJ
      IS=KPJ-NX
      IF(IS.LT.1)IS=1
      SUM=0
      DO 95 I=IS,K-1
95    SUM=SUM+AU(I,K-I)*AU(I,KPJ-I)/AU(I,0)
```

```
      AU(K,JJ)=AAS(K,J)-SUM
94    CONTINUE
91    CONTINUE
C*********************************************
      AUINV(1,0)=1.0
      DO 534 K=NACT,2,-1
      KM1=K-1
      AUINV(K,0)=1.0
      AUINV(KM1,1)=AUINV(KM1,1)-AU(KM1,1)/AU(K,0)
      DO 774 KK=0,1
774   AUINV(K,KK)=AUINV(K,KK)/AU(K,0)
534   CONTINUE
C*************************************************
C     SOLVE FOR PRESSURES
      NTRIAL=500
      ERROR=0.0001
      BETA=0
      NACTM1=NACT-1
      DO 451 K=1,NTRIAL
      RMAX=0.0
      ALINR(1)=AUINV(1,0)*R(1)*AU(1,0)
      DO 462 N=2,NACT
      NM1=N-1
462   ALINR(N)=(AUINV(N,0)*R(N)+
     *          AUINV(NM1,1)*R(NM1))*AU(N,0)
      SUM1=0
      DO 961 N=1,NACTM1
      AUINR(N)=AUINV(N,0)*ALINR(N)+
     *          AUINV(N,1)*ALINR(N+1)
961   SUM1=R(N)*AUINR(N)+SUM1
      AUINR(NACT)=AUINV(NACT,0)*ALINR(N)
      SUM1=R(NACT)*AUINR(NACT)+SUM1
      IF(K.GT.1)BETA=SUM1/SUM3
      DO 4661 N=1,NACT
4661  SS(N)=BETA*SS(N) +AUINR(N)
      SUM2=0
      DO 4991 N=1,NACT
      SUM=0
      DO 192  L=1,4
      J=LOC(N,L)
192   IF(J.NE.0)SUM=AAS(N,L)*SS(N+J)+SUM
      ALFSUM(N)=SUM+AAS(N,0)*SS(N)
4991  SUM2=SS(N)*ALFSUM(N)+SUM2
      ALFA=SUM1/SUM2
      SUM3=SUM1
      DO 4681 N=1,NACT
4681  R(N)=R(N)-ALFSUM(N)*ALFA
      DO 9671 N=1,NACT
      PSAVE(N)=PSAVE(N)+ALFA*SS(N)
      R2=ABS((PSAVE(N)-PSAVE2(N))/PSAVE(N))
      IF(R2.GT.RMAX)RMAX=R2
9671  PSAVE2(N)=PSAVE(N)
      IF(RMAX.LE.ERROR)GO TO 100
451   CONTINUE
```

```
C**********************************************
100     DO 9 M=1,NACT
        N=INDEX(M)
9       CALP(N)=PSAVE(M)
        RETURN
        END
```

```
C
C     THE NF METHOD
C
      SUBROUTINE NF(A,B,C,E,F,D,CALP,NX,NY,NXNY,NUM2)
      REAL*8  D(NXNY),AMI(361),SUM,VAL,R(361),RMAX,AM,
     *  T(19,-1:1),TINV(19,19),COLSUM(19),CALP(NXNY),
     * W(-19:380),PSAVE(-19:380),E(NXNY),A(NXNY),B(NXNY),C(NXNY),
     * F(NXNY),DELP(-19:380),PSAVE2(-19:380)
      INTEGER*4 NX,NY,NXNY
      IF(NFLAG.EQ.1)GO TO 12
      NFLAG=1
      DO 36 N=-NX,NXNY+NX
      PSAVE(N)=0.0
      PSAVE2(N)=0.0
      W(N)=0.0
36    DELP(N)=0.0
12    DO 95 N=1,NX
95    COLSUM(N)=0.0
      DO 35 N=1,NXNY
35    R(N)=D(N)
      NUM=0
      DO 134 J=1,NY
      SUM=0
      DO 135 I=1,NX
      NUM=NUM+1
      ASUM=A(NUM)*SUM
      AM=B(NUM)-ASUM-COLSUM(I)
      T(I,0)=AM+ASUM
      T(I,1)=C(NUM)
      T(I,-1)=A(NUM)
      AMI(NUM)=1/AM
135   SUM=AMI(NUM)*C(NUM)
C CALCULATE INVERSE OF T(N,N) MATRIX
      IF(NUM.EQ.NXNY)GO TO 134
      DO 159 N=1,NX
      DO 158 K=1,NX
158   TINV(N,K)=0.0
159   TINV(N,N)=1.0
      DO 51 K=1,NX-1
      I=K+1
      VAL=T(I,-1)/T(K,0)
      T(I,0)=T(I,0)-VAL*T(K,1)
      DO 152 JJ=1,K
152   TINV(I,JJ)=TINV(I,JJ)-VAL*TINV(K,JJ)
51    CONTINUE
      DO 4590 K=NX,2,-1
      KM1=K-1
      VAL=T(KM1,1)/T(K,0)
      DO 4590 JJ=NX,1,-1
4590  TINV(KM1,JJ)=TINV(KM1,JJ)-VAL*TINV(K,JJ)
```

```
      DO 7940 N=1,NX
      DO 7940 KK=1,NX
7940  TINV(N,KK)=TINV(N,KK)/T(N,0)
      N2=NUM-NX
      DO 7760 K=1,NX
      NUMPK=NUM+K
      DO 7760 K2=1,NX
7760  TINV(K,K2)=E(NUMPK)*TINV(K,K2)
      DO 7761 K=1,NX
      N2PK=N2+K
      DO 7761 K2=1,NX
7761  TINV(K2,K)=TINV(K2,K)*F(N2PK)
      DO 5930 N=1,NX
      COLSUM(N)=0.0
      DO 5930 K=1,NX
5930  COLSUM(N)=TINV(K,N)+COLSUM(N)
134   CONTINUE
C  SOLVE FOR PRESSURES
      NXM1=NX-1
      NTRIAL=200
      ERROR=0.0001
121   DO 451 K=1,NTRIAL
      RMAX=0.0
      DO 23 J=1,NY
      N1=J*NX
      N2=N1-NXM1
      DO 449 I=N2,N1
449   DELP(I)=AMI(I)*(R(I)-E(I)*DELP(I-NX)-A(I)*DELP(I-1))
      DO 453 I=N1,N2,-1
453   DELP(I)=DELP(I)-AMI(I)*C(I)*DELP(I+1)
23    CONTINUE
      DO 24 J=NY,1,-1
      N1=J*NX
      N2=N1-NXM1
      DO 455 I=N1,N2,-1
455   W(I)=AMI(I)*(F(I)*DELP(I+NX)-C(I)*W(I+1))
      DO 456 I=N2,N1
      W(I)=W(I)-AMI(I)*(A(I)*W(I-1)-F(I)*W(I+NX))
      DELP(I)=DELP(I)-W(I)
456   PSAVE(I)=PSAVE(I)+DELP(I)
24    CONTINUE
      DO 234 N=1,NXNY
      R(N)=D(N)-A(N)*PSAVE(N-1)-
     *     E(N)*PSAVE(N-NX)-
     *     B(N)*PSAVE(N)-
     *     C(N)*PSAVE(N+1)-
     *     F(N)*PSAVE(N+NX)
      R2=ABS((PSAVE(N)-PSAVE2(N))/PSAVE(N))
      IF(R2.GT.RMAX)RMAX=R2
234   PSAVE2(N)=PSAVE(N)
      IF(RMAX.LE.ERROR)GO TO 100
451   CONTINUE
      NTRIAL=200
      GO TO 121
```

```
100     DO 244 N=1,NXNY
244     CALP(N)=PSAVE(N)
        RETURN
        END
```